AD-A120 378 SYSTEMS ARCHITECTS INC RANDOLPH MASS F/G 9/2
COMPUTER SYSTEMS ACQUISITION METRICS HANDBOOK. VOLUME IV. THEOR--ETC(U)
MAY 82 F19628-80-C-0207
UNCLASSIFIED ESD-TR-82-143(4) NL

1 of 1

END
DATE
FILMED
11 82

ESD-TR-82-143(IV)

AD A120378

THEORETICAL SUPPLEMENT FOR COMPUTER SYSTEMS ACQUISITION METRICS HANDBOOK. VOLUME IV.

Systems Architects, Inc.
50 Thomas Patten Drive
Randolph, MA 02368

May 1982

Approved for public release;
Distribution Unlimited.

Prepared for

ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
DEPUTY FOR TECHNICAL OPERATIONS AND
PRODUCT ASSURANCE
HANSCOM AIR FORCE BASE, MASSACHUSETTS 01731

LEGAL NOTICE

OTHER NOTICES

REVIEW AND APPROVAL

This technical report has been reviewed and is approved for publication.

Robert V. Vieraitis Jr.

ROBERT V. VIERAITIS, Jr., 1Lt, USAF
Project Officer

James W. Neely, Jr.

JAMES W. NEELY, Jr., Lt Col, USAF
Chief, Computer Engineering
Applications Division

FOR THE COMMANDER

Walter W. Turgiss

WALTER W. TURGISS
Acting Director, Engineering and Test
Deputy for Technical Operations
and Product Assurance

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: ESD-TR-82-143(IV)	2. GOVT ACCESSION NO.: AD-A120378
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): Theoretical Supplement for Computer Systems Acquisition Metrics Handbook. Volume IV.	5. TYPE OF REPORT & PERIOD COVERED
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Systems Architects, Inc.	8. CONTRACT OR GRANT NUMBER(s): F19628-80-C-0207
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Systems Architects, Inc 50 Thomas Patten Drive Randolph, MA 02368	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Electronic Systems Division (TOEE) Hanscom AFB Massachusetts 01731	12. REPORT DATE: May 1982
	13. NUMBER OF PAGES: 78
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer systems
Metrics
Quality assurance
Software

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This volume documents the background research and describes the various metrics approaches that were analyzed. It goes on to describe the methodology selected and defines the framework for the handbook.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

SECTION I

INTRODUCTION AND BACKGROUND

1.1 NEED FOR SQA MEASUREMENTS

The Air Force perceives computer systems as an increasing cost which must be harnessed and controlled. Computer systems are rapidly increasing in importance as components of management and defense systems. The Air Force is responsible for the selection, development and maintenance of numerous computer systems. Due to a lack of quantifiable criteria to judge the quality of a computer system during its life cycle and to other causes, the acquisition of computer systems has resulted in costly problems for the Air Force. These problems often are not observed until after the system has been developed, and may include the following:

- Failure to meet user specifications,
- Lack of reliability in performing intended functions,
- Inefficient use of computing resource and code,
- Failure to meet access control requirement,
- Difficult to use,
- Difficult to modify or upgrade,
- Difficult to transfer from one system environment to another,
- Unable to be used in other applications, and
- Unable to interface with other systems.

Methods have been developed to prevent or lessen the severity of these problems. These methods include training programs for software acquisition personnel, documentation of past experience, and procedures and tools to assure the quality of developed systems. Guidelines and procedures have been provided in a series of software acquisition management guidebooks. These guidebooks cover configuration management, computer program development specification, documentation requirements, verification, validation and certification, software maintenance, software quality assurance, software cost estimation and measurement, software development and maintenance facilities, and life cycle events.

These guidebooks provide qualitative methods for the assessment of software development. Quantitative measures, or metrics, have recently been developed as a tool to quantitatively measure the quality of software. These metrics should provide an additional means to assess the quality of the software, and to reduce or prevent the problems listed above.

As a part of its software quality program, the Air Force Systems Command (AFSC), Electronic Systems Division (ESD) has contracted Systems Architects, Inc. (SAI) to develop a "Computer Systems Acquisition Metrics Handbook". This Handbook will enable ESD personnel to apply software quality metrics to software development efforts. The purpose of this theoretical supplement is to demonstrate the work done producing this "Computer Systems Acquisition Metrics Handbook".

The research in the initial phase of this project indicated that of the work that had been done in the area of software metrics, the metrics developed by McCall at General Electric (GE) (RADC-TR-80-109, Vol. I & II) were most applicable to ESD's environment. These metrics were developed in the Department of Defense (DoD) environment. These metrics were adapted to form the theoretical basis for the development of this "Handbook".

1.2 SOFTWARE METRICS LITERATURE REVIEW

In order to provide a comprehensive review of software metrics and related areas of research, SAI conducted a literature review. This review met three objectives: (1) Outline state-of-the-art in software metric efforts; (2) Provide a scope of the problems involved in software development and acquisition; and (3) Determine what solutions have been found to these problems for incorporation into this effort where relevant.

The literature review identified a number of factors contributing to the poor development of software, and some theories and approaches developed to exert better control and produce more reliable software. The areas of current literature investigated included metrics, software reliability, software management, software quality tools and techniques, and software cost estimation.

Software metrics are measures used to evaluate the quality of software over its life cycle. It is currently an infant discipline, and there are a number of conflicting opinions as to what and how software characteristics should be measured. Some of the theories and methods that were found for measuring software characteristics include the following:

- Boehm, Brown, and Lipow's Quality Characteristics Tree,
- Gilb's Software Metrics,
- Halstead's Software Science,
- McCall's Metrics,
- McCabe's Complexity Measure,
- Measures of Comprehensibility

These metrics were in the validation stage when the early efforts toward developing the "handbook" took place. The major directions in metrics research was: (1) The identification of metrics related to a specific software attribute; and (2) The use of metrics to evaluate programmer performance or human factors involved in software developments. Areas of future research are software management and automated measurement. Considerable benefits can are are being realized from this research.

Software reliability has been defined as the degree to which a software system satisfies its requirements, and delivers usable services. There are several different types of software reliability models, operational reliability models, and user-oriented models. Although a standard cookbook approach for widespread application of these models cannot now be provided, software reliability measurement have clearly progressed beyond the pure theory stage. The tools that have been developed can be useful and valuable in software development efforts.

Software management includes the estimation, negotiation, and control of a technology which is very complex and which is subject to a high degree of modification. Typically, the following four problem areas are encountered:

(1) Obtaining satisfactory software requirements,

(2) Improving the art of software cost estimating,

(3) Achieving significant productivity improvements, and

(4) Maintaining control and visibility of software developments.

Obtaining satisfactory software requirement specification is the first obstacle to the success of software projects. English language statements are currently the major form of specification. Other forms of specification (including automated tools and simulation) have been advocated. Productivity can be improved by providing software designers with automated tools. The key to effective control is to break up the development of software into a number of small measurable steps, and then to audit the satisfactory completion of those steps. Techniques necessary for controlling the development of large software based systems include: stepwise refinement, requirements traceability, process design, incremental development, structured development, software design language, unit development folders, quality assurance/configuration management, and life cycle maintenance.

There is a wide range of tools and techniques for improving software quality over the life cycle. Specific tools and techniques have been developed for software design, implementation, checkout, and maintenance. Some of the tools and techniques have been applied to software development projects economically and with successful results, but many are still in Research and Development.

Software cost estimating is considered an imprecise art. At the start of this project the best single indicator of software development cost was the number of machine instructions. Using this number has yielded high amounts of estimating errors. A number of qualitative factors are needed to provide a better picture of the true potential cost. Two cost estimation models which have been developed include the RCA PRICE-S and the TRW SCEP. Both of these models have been applied to a number of software development projects.

Progress in the advancement of software technology and tools over the past decade has been rapid, but developments in software have not kept pace with the potential for development provided by hardware innovations. The tools and techniques described above require further validation and refinement in order to provide for the development of software that is economical, reliable, and functional.

1.3 CONCEPTS AND CLASSIFICATION OF SOFTWARE METRICS

1.3.1 Concepts of Software Metrics

Software metrics is currently an infant discipline, and there are conflicting opinions as to what and how software characteristics should be measured (SHNB80)*. R. Rubey and R. Hartwick first introduced the concept of software metrics in 1968. Since that time, the research in software metrics has grown to include (MCCJ80c)*:

- The use of metrics as an aid in testing and maintaining software,
- The psychological and programmer performance implications,
- The use of metrics as software acceptance criteria in a formal acquisition environment, and
- The use of metrics in providing a quality assurance tool/technique.

The major proposed methods for measuring computer program quality are described in the following subsections. These proposed methods for software metrics include:

*Key to reference listed in Appendix A.

(1) Boehm, Brown and Lipow's Quality Characteristics Tree,

(2) Gilb's Software Metrics,

(3) Halstead's Software Science,

(4) McCall's Metrics,

(5) McCabe's Complexity Measure, and

(6) Measures of Comprehensibility.

1.3.1.1 <u>Boehm, Brown and Lipow's Quality Characteristic Tree</u>

In determining what characteristics should be measured in order to determine software quality, Boehm, Brown and Lipow (1978) defined a hierarchical software quality characteristic tree. In the quality characteristic tree (Figure I-1), the arrow implies that the quality to the left of the arrow must also include the quality/qualities to the right of the arrow. Thus, a program that is maintainable must also be testable, understandable, and modifiable (SHNB80). The higher levels of the system reflect the uses of the software quality evaluation in terms of:

(1) The current behavior of the software,

(2) The ease of changing the software, and

(3) The ease of converting or interfacing the software system (CURB80b).

The lowest level characteristics (right-most in Figure I-1) are "primitives", or the most basic characteristics. These lowest level characteristics may be combined into the medium characteristics and are recommended as software metrics for both the

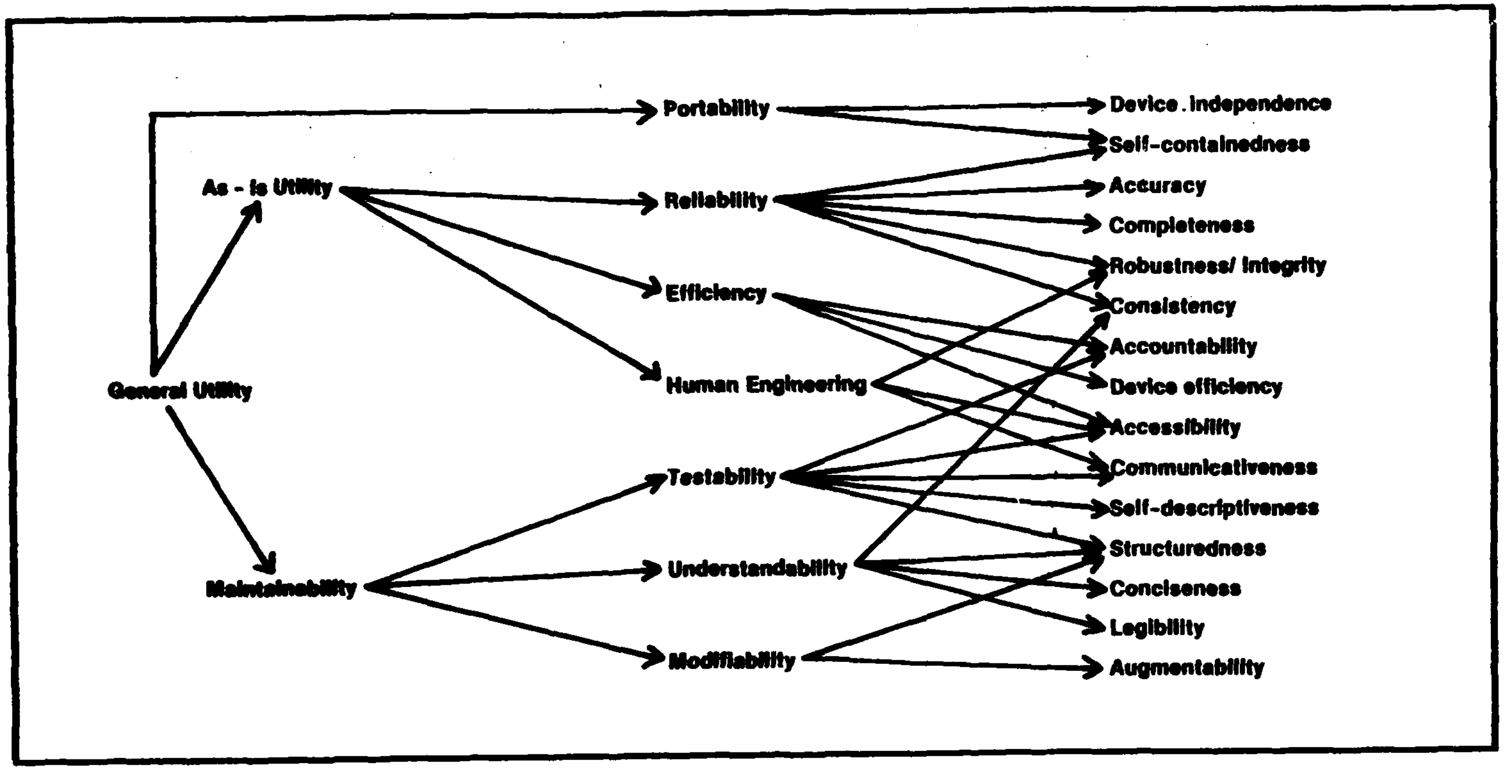


FIGURE I-1

SOFTWARE QUALITY CHARACTERISTIC TREE

"primitives" and the higher level characteristics. The detailed and complex scheme proposed by Boehm, Brown and Lipow is based on practical experience and is appealing to programmers. However, Boehm, Brown and Lipow offer no clear cut demonstration of its effectiveness, reliability, or applicability in other environments. The lengthy list of issues acts as a checklist for reviewing a program rather than offering guidance in program construction. Such checklists can be useful, but they tend to grow long and cumbersome, match the needs of a specific organization, and become language/system specific [SHNB80].

1.3.1.2 Gilb's Software Metrics

Gilb (1977) presents a set of basic software metrics, making no claim as to their completeness. He emphasizes that each software application requires its own concepts and that his text is intended to provide basic concepts on which the user can build. Gilb builds a strong and convincing case for a precise measurement based on the history of the physical sciences. Gilb's major categories of metrics include: reliability metrics, flexibility metrics, structure metrics, performance metrics, resource metrics, and diverse metrics [GILB77]. Each of these categories is broken down into lists of subcategories. Many of Gilb's metrics are difficult to obtain. Even where values can be computed, there exists no sense of the range of good values. The lack of independence of the metrics adds to the confusing complexity and makes it difficult for programmers to predict the effect of a program change on a group of metrics [SHNB80].

1.3.1.3 Halstead's Software Science

Halstead (1977) approaches the human preparation of computer programs by using methods and principles of classical experimental science.

This software science is based on counting the operators and operands of any program. This is a difficult count to make for large programs unless automated. An operator is defined as the what-to-do portion of a program instruction, and an operand is a peice of data upon which an operation is performed. From these, Halstead derives mathematical formulas to determine estimates of:

- Program size,
- Programming time,
- The initial number of errors to be expected in a program, and
- The number of errors delivered in a program.

Using these formulas, Halstead has derived a program modularization scheme. Programs modularized according to this scheme, says Halstead, will be easy to write, debug, comprehend, and maintain. Halstead's software science is appealing, the experimental evidence is convincing, and its psychological foundations are sturdy. However, Halstead's approach is not complete since it ignores specific issues such as complexity and choice of algorithms, and general issues such as portability, flexibility, and efficiency [SHNB80]. Halstead's theory has been the subject of considerable research, and results indicate that Halstead's metrics are in fact good indicators of the number of bugs in a program, programming time, and debugging time [CURB80b].

1.3.1.4 McCall's Metrics

Research sponsored by the U.S. Air Force led to McCall's software metrics model. This model contains a comprehensive hierarchical definition of software quality (see Figure I-2). At the highest level, quality factors are defined that are appropriate for software acquisition managers to use as an aid in specifying quality objectives for their software systems. These high level factors are

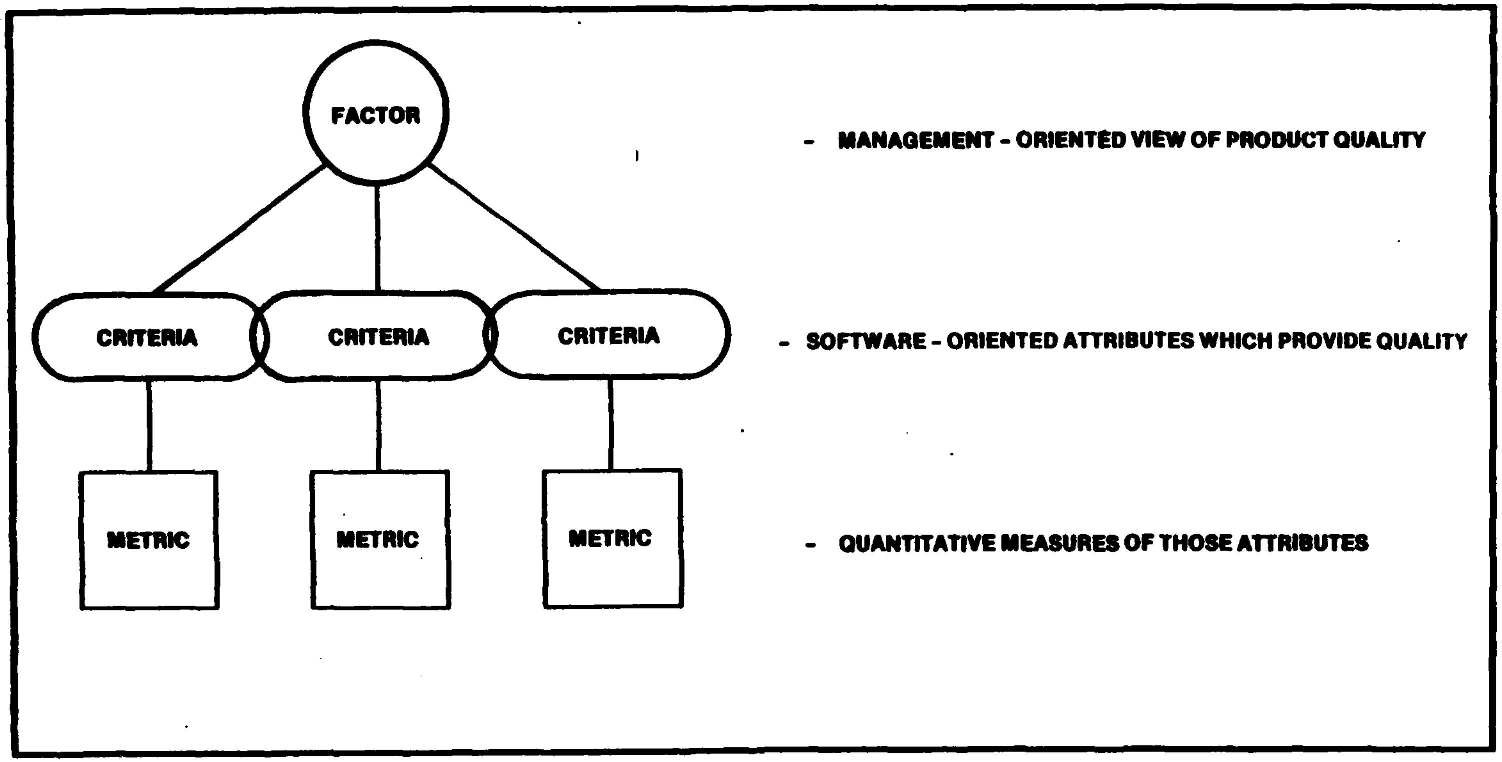


FIGURE I-2

SOFTWARE QUALITY FRAMEWORK

more software-directed until specific metrics are proposed that relate to the factors [CAVJ78].

The application of the framework for McCall's metrics has three impacts on quality assurance activities during large-scale software development. These impacts [MCCJ80v] are:

(1) The framework provides a mechanism for a program manager to identify what qualities are important.

(2) The framework provides a means of quantitatively assessing how well the development is progressing relative to the quality goals established.

(3) The framework provides for more interaction by the quality assurance personnel throughout the development effort.

McCall's metrics have been documented in the RADC Software Quality Measurement Manual. On-going work involves automating the function of data collection for the metrics [MCCJ80].

1.3.1.5 McCabe's Complexity Measure

McCabe asserts that half of software development is spent in testing, and most of the money spent on software systems is used for maintaining the system. Therefore, what is needed is a mathematical technique that provides a quantitative basis for modularization and identifies software modules which will be difficult to test or to maintain [MCCT76].

McCabe (1976) describes a complexity measure and illustrates its use in managing, testing, and controlling program complexity. He defines this complexity measure as the number of paths through a program. McCabe's theory assumes that the complexity of a program is not dependent on its size and that the complexity of a program depends only on the structure of the decisions in the program. McCabe says that merely restricting the size of a program module does not ensure good modularization and points out that it is possible for a fifty-line module to have 33.5 million distinct control paths. His approach is to measure and control the number of paths through a program.

McCabe's complexity measure is a helpful tool in preparing test data and may provide useful information related to program complexity. However, his metric ignores the choice of algorithms and how data is structured, and avoids important considerations such as portability, flexibility, and efficiency [SHNB80].

1.3.1.6 Measures of Comprehensibility

Comprehensibility is the ease with which a programmer can understand a program. Sheppard, Borst and Love (1978) conducted an experiment to evaluate the effect of structure on a programmer's understanding of a computer program, and the use of Halstead's and McCabe's metrics on the prediction of program understanding.

In the experiment, professional programmers were given ten minutes to study a short program and five minutes to reconstruct an equivalent program. The experimental results indicated that the least structured program was the most difficult to reconstruct and a partially structured program was the easiest. McCabe's complexity measure was found to be a good predictor of program length, as was Halstead's E metric (the amount of effort required to generate a program).

Schneiderman (1978) has proposed that every program module should meet a 90-10 rule: "A competent programmer should be able to reconstruct 90 percent of the program after 10 minutes of study." This concept, which is based on several memorization/reconstruction experiments, needs testing and validation [SHNB80].

1.3.1.7 Summary

Two major directions can be identified in the area of software metric research. In one direction, researchers are identifying metrics related to a specific attribute of a software product's quality. The best examples of this type of metric are the complexity measures that have been derived. The other direction has been to use metrics for the evaluation of programmer performance or the human factors involved in software developments. Research efforts will continue in the area of the psychological aspects of software development and in specialized studies of various metrics for each quality factor.

Other efforts will continue in the area of providing automated measurement, which is required to make the application of metrics economical and consistently reliable. Another area that future research will attach is that of software management, where metrics will provide a feedback mechanism for viewing software development as a controlled process. Considerable benefits can and are currently being realized from the research.

SECTION II

GE METRIC EFFORT

2.1 SCOPE OF GE METRICS

Research sponsored by the U.S. Air Force Electronics Systems Division (ESD) and Rome Air Development Center (RADC) led to the development of a concept of software quality by the General Electric Company (GE). One of the objectives of this effort was to provide Air Force system acquisition managers with a mechanism to quantitatively specify and measure the desired level of quality in a software product. McCall at GE developed a system of measures, or metrics, for the quantitative specification and measurement of software quality. The concepts, definitions, and classification of these metrics are provided in the subsections which follow.

2.1.1 Framework

The software metrics framework is a hierarchical structure. Management-oriented Quality Factors at the highest level are important to software acquisition managers and used as an aid in specifying quality objectives for their software systems. At the next level are Criteria which are software-oriented attributes used to describe characteristics of the software. At the next level are Metrics which provide a measurement of the software Criteria. This hierarchical framework is presented in Figure II-1.

The measurements are to be taken during the development effort. These measurements are not post-implementation assessments of software quality. They are not test-like measurements. Their purpose is to provide an indication of the progress toward a desired level of quality during the development. The Criteria, established for each software Quality Factor, represent attributes which can be measured during the software development. A detailed description of each of the levels in this framework is provided in the following subsections.

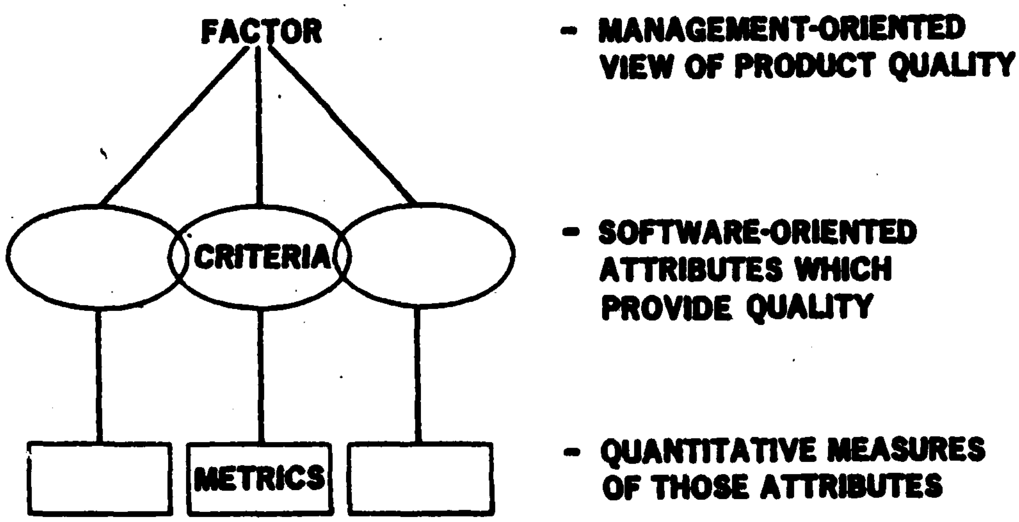


FIGURE II-1

METRICS HIERARCHICAL FRAMEWORK

2.1.1.1 Quality Factors

Quality Facotrs are management oriented software qualities used to identify what qualities are desired in the software product being developed. This metric system uses the following eleven software Quality Factors.

(1) Correctness - the extent to which a program satisfies its specifications and fulfills the user's mission objectives,

(2) Reliability - the extent to which a program can be expected to perform its intended function with required precision,

(3) Efficiency - the amount of computing resources and code required by a program to perform a function,

(4) Integrity - the extent to which access to software or data by unauthorized persons can be controlled,

(5) Usability - the effort required to learn, operate, prepare input, and interpret output of a program,

(6) Maintainability - the effort required to locate and fix an error in an operational program,

(7) Testability - the effort required to test a program to insure it performs its intended function,

(8) Flexibility - the effort required to modify an operational program,

(9) Portability - the effort required to transfer a program from one hardware configuration and/or software system environment to another.

(10) Reusability - the extent to which a program can be used in other applications (related to the packaging and scope of the function that programs perform), and

(11) Interoperability - the effort required to couple one system with another.

These Factors provide a mechanism to specify the basic attributes of a system over its life cycle. Thus, if a system is being developed in an environment where there is a high rate of technological breakthroughs in hardware design, portability should be given a primary significance. If the expected life cycle of the system is long, then maintainability becomes a cost-critical consideration. If the system is an experimental system where the software specifications will have a high rate of change, flexibility in the software product is highly desirable. If the functions of the system are expected to be required for a long time, while the system itself may change considerably from time to time, then reusability is highly significant for those modules which implement the major functions of the system. The quality of interoperability becomes extremely important for systems required to interface with others via communications networks.

2.1.1.2 Criteria

Software Criteria are attributes of the software and further define the hierarchical structure of GE's metrics. Each Factor is defined by a set of Criteria; Criteria which affect more than one Factor help to describe the relationships between Factors. The software Criteria are listed and defined in Table II-A. The relationship between Criteria and Factors is shown in Table II-B.

2.1.1.3 Metrics

The actual measurement of software quality is accomplished by applying Software Metrics to the documentation and source code produced during a software development effort. To determine the value of a metric questions are asked that are answerable numerically or by Yes or No response. The Yes or No responses are translated to 1 and 0 respectively so that in effect all answers are numerical. Metrics are established in a one to one relationship with criteria to provide a measure of the software Criteria. Which in turn combine through algorithms to Quantify the Quality Factor.

The metrics are organized using five phases of software development: (1) Requirements Analysis (2) Preliminary Design, (3) Detail Design, (4) Implementation, and (5) Test and Integration. See Figure II-2.

CRITERIA	DEFINITION
Traceability	Those attributes of the software that provide a thread from the requirements to the implementation, with respect to the specific development and operational environment.
Completeness	Those attributes of the software that provide full implementation of the functions required.
Consistency	Those attributes of the software that provide uniform design and implementation techniques and notation.
Accuracy	Those attributes of the software that provide the required precision in calculations and outputs.
Error Tolerance	Those attributes of the software that provide continuity of operation under nonnominal conditions.
Simplicity	Those attributes of the software that provide implementation of functions in the most understandable manner. (Usually avoidance of practices which increase complexity.)
Modularity	Those attributes of the software that provide a structure of highly independent modules.
Generality	Those attributes of the software that provide breadth to the functions performed.

TABLE II-A

SOFTWARE CRITERIA

CRITERIA	DEFINITION
Expandability	Those attributes of the software that provide for expansion of data storage requirements or computational functions.
Instrumentation	Those attributes of the software that provide for the measurement of usage or identification of errors.
Self-Descriptiveness	Those attributes of the software that provide explanation of the implementation of a function.
Execution Efficiency	Those attributes of the software that provide for minimum processing time.
Storage Efficiency	Those attributes of the software that provide for minimum storage requirements during operation.
Access Control	Those attributes of the software that provide for control of the access of software and data.
Access Audit	Those attributes of the software that provide for an audit of the access of software and data.
Operability	Those attributes of the software that determine operation and procedures concerned with the operation of the software.

TABLE II-A (continued)

SOFTWARE CRITERIA

CRITERIA	DEFINITION
Training	Those attributes of the software that provide transition from current operation or initial familiarization.
Communicativeness	Those attributes of the software that provide useful inputs and outputs which can be assimilated.
Software System Independence	Those attributes of the software that determine its dependency on the software environment (operating systems, utilities, input/output routines, etc.).
Machine Independence	Those attributes of the software that determine its dependency on the hardware system.
Communications Commonality	Those attributes of the software that provide the use of standard protocols and interface routines.
Data Commonality	Those attributes of the software that provide the use of standard data representations.
Conciseness	Those attributes of the software that provide for implementation of a function with a minimum amount of code.

TABLE II-A (continued)

SOFTWARE CRITERIA

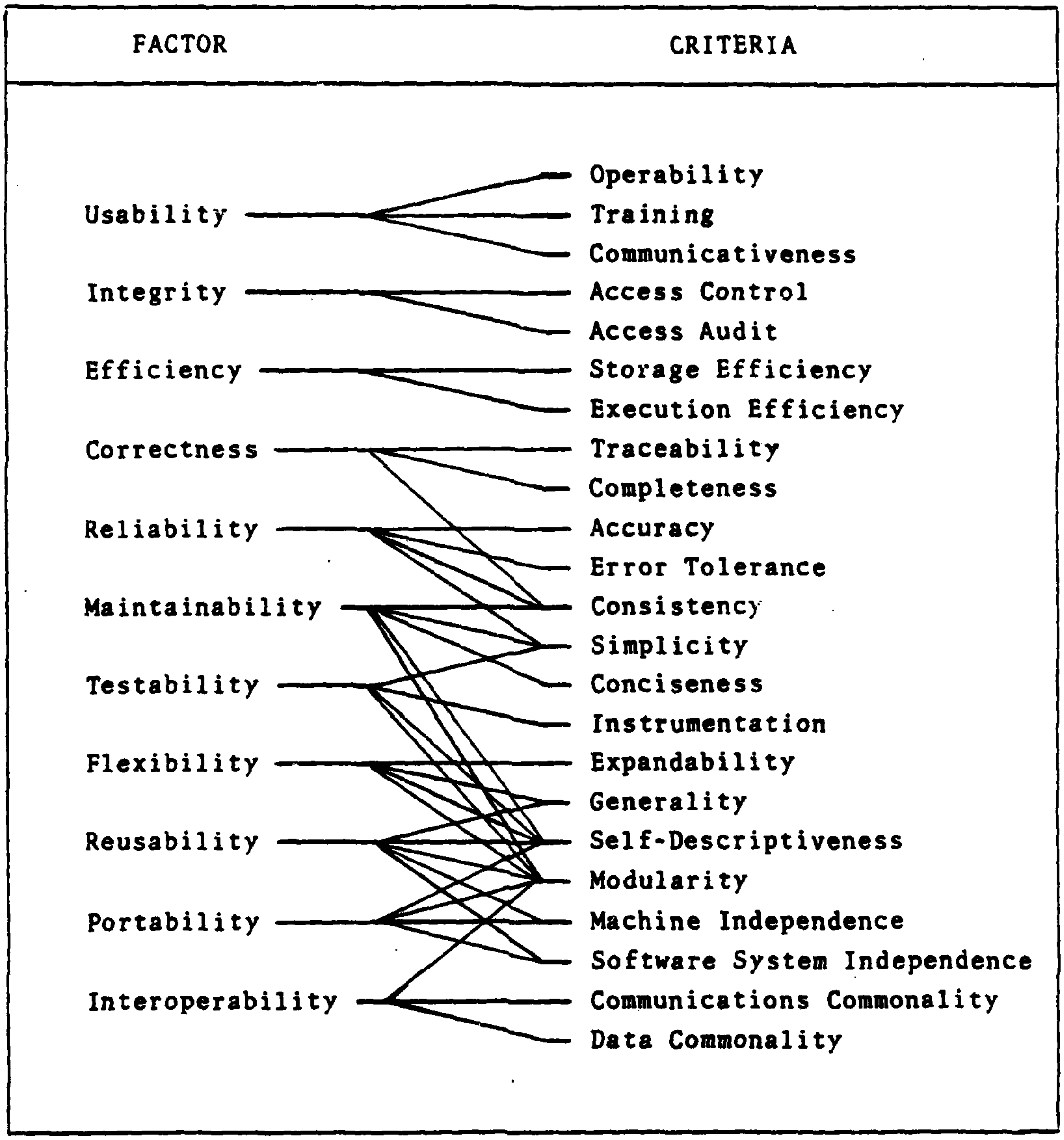


TABLE II-B

RELATIONSHIP OF FACTORS AND CRITERIA

The metrics can be applied either at the system level or subsystem level if the subsystem is viewed as the "System". At the system level, the metrics can be utilized to obtain an overall measure of how the system is progressing with respect to a particular Quality Factor. At the subsystem level, the metrics can be used to identify problems in a particular subsystem so that corrective actions or an emphasis can be applied to that subsystem. Appendix B of GE's Software Quality Metrics Enhancements (RADC-TR-80-109) provide a description and explanation of the use of GE's metrics.

2.2 GE PROCEDURE FOR APPLYING METRICS

The procedure used to apply GE's metrics involves three steps: (1) Identify software quality requirements, (2) Apply software quality measurements, and (3) Assess the quality of the software product. The procedures involved in these steps are described in the following subsections.

2.2.1 Identify Software Quality Requirements

Activities in identifying software quality requirements include identifying important Quality Factors, identifying critical software attributes, and establishing quantifiable goals. To identify software quality requirements, a survey form is used to solicit responses from the system's decision makers (acquisition manager, user/customer, development manager, and quality assurance manager).

This survey form, shown in Figure II-3, is used to gather information with respect to the basic characteristics of the application. This identifies the rank of Quality Factors, and documents the rationale for the decisions made in selecting the Quality Factors.

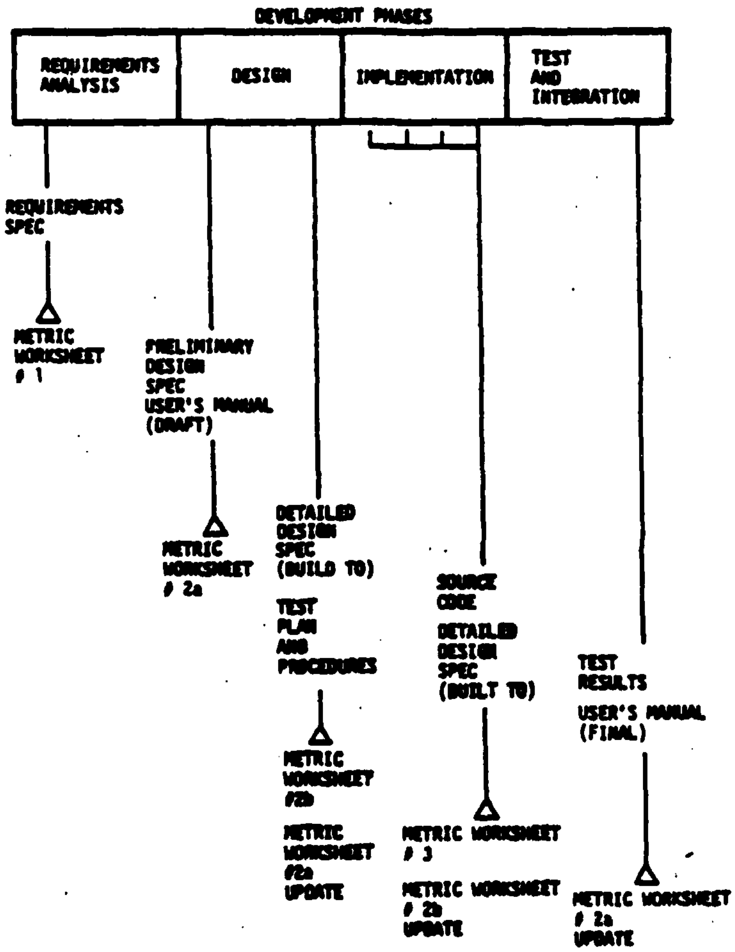


FIGURE II-2

TIMING OF THE APPLICATION OF THE METRIC WORKSHEETS

1. The 11 quality factors listed below have been isolated from the current literature. They are not meant to be exhaustive, but to reflect what is currently thought to be important. Please indicate whether you consider each factor to be Very Important (VI), Important (I), Somewhat Important (SI), or Not Important (NI) as design goals in the system you are currently working on.

RESPONSE	FACTORS	DEFINITION
______	CORRECTNESS	Extent to which a program satisfies its specifications and fulfills the user's mission objectives.
______	RELIABILITY	Extent to which a program can be expected to perform its intended function with required precision.
______	EFFICIENCY	The amount of computing resources and code required by a program to perform a function.
______	INTEGRITY	Extent to which access to software or data by unauthorized persons can be controlled.
______	USABILITY	Effort required to learn, operate, prepare input, and interpret output of a program.
______	MAINTAINABILITY	Effort required to locate and fix an error in an operational program.
______	TESTABILITY	Effort required to test a program to insure it performs its intended function.
______	FLEXIBILITY	Effort required to modify an operational program.
______	PORTABILITY	Effort required to transfer a program from one hardware configuration and/or software system environment to another.
______	REUSABILITY	Extent to which a program can be used in other applications - related to the packaging and scope of the functions that programs perform.
______	INTEROPERABILITY	Effort required to couple one system with another.

2. What type(s) of application are you currently involved in?

____

3. Are you currently in:

______ 1. Development phase
______ 2. Operations/Maintenance phase

4. Please indicate the title which most closely describes your position:

______ 1. Program Manager
______ 2. Technical Consultant
______ 3. Systems Analyst
______ 4. Other (please specify)______

FIGURE II-3

SOFTWARE QUALITY REQUIREMENTS SURVEY FORM

The degree of Relationship between software Quality Factors is shown in Figure II-4. This is used to "trade-off" Quality Factors selection so that Quality Factors with a low degree of relationships are not expected to exist simultaneously within a system.

The next level of identifying the quality measurements is to proceed from the management-oriented Quality Factors to the software-oriented Criteria. The Criteria are related to the various factors by definition and provide a more detailed specification of the quality requirements.

After the critical quality factors have been identified, specific performance levels or ratings required for each factor should be specified. The specific metrics which will be applied to the various software products produced during the development should be specified, and specific minimum values for particular metrics may be specified in addition to the ratings.

2.2.2 Apply Software Quality Measurements

The vehicle for applying the software quality measurements are the metric worksheets contained in GE's Software Quality Metrics Manual (RADC-TR-80-109). The metric worksheets are applied to the available system documentation or source code and the measurements are translated into metric scores. The application of the metric worksheets follow the phased development of the software. The timing of the application of the metric worksheets is shown in Figure II-2.

2.3 INTERPRETATION OF GE METRICS

The benefits of applying the software quality metrics are realized when the information gathered from the application of the metric worksheets is analyzed. There are three levels at which analyses can be performed: (1) Inspector's Assessment, (2) Sensitivity Analysis, and (3) Use of Normalization Function. In the Inspector's Assessment, an inspector, using the worksheets, asks

FACTORS	CORRECTNESS	RELIABILITY	EFFICIENCY	INTEGRITY	USABILITY	MAINTAINABILITY	TESTABILITY	FLEXIBILITY	PORTABILITY	REUSABILITY	INTEROPERABILITY
CORRECTNESS											
RELIABILITY	○										
EFFICIENCY											
INTEGRITY			●								
USABILITY	○	○	●	○							
MAINTAINABILITY	○	○	●		○						
TESTABILITY	○	○	●		○	○					
FLEXIBILITY	○	○	●	●	○	○	○				
PORTABILITY			●			○	○				
REUSABILITY		●	●	●		○	○	○	○		
INTEROPERABILITY			●	●					○		

LEGEND

If a high degree of quality is present for factor, what degree of quality is expected for the other:

○ = High ● = Low

Blank = No relationship or application dependent

FIGURE II-4

RELATIONSHIPS BETWEEN SOFTWARE QUALITY FACTORS the same questions and takes the same counts for each module's source code or design document that is reviewed. Based on this consistent evaluation, a subjective comparison of the products can be made.

The Sensitivity Analysis uses the results of the calculations from the metric worksheets to form a matrix of measurements. The matrix represents a profile of all the modules in the system with respect to a number of characteristics measured by the metrics, and allows a number of analyses to be performed. The last level of quality assessment involves using normalization functions to predict the quality of the software in quantitative terms.

The majority of the coefficients required to perform the sensitivity analysis are not available at this time due to their empirical nature. More research and development must be done in this area to provide full information at all three levels of interpretation.

SECTION III

METHODOLOGY FOR TRANSFORMING THE GE METRICS INTO THE SOFTWARE METRICS HANDBOOK

3.1 OBJECTIVES

The objectives of this project were to develop a standard set of procedures that quantitatively specify and measure the quality of a software system during its life cycle. The procedures were written so that they could be learned during a one week training program by students with limited software system development knowledge. These procedures are documented in the form of a "Computer Systems Acquisition Metrics Handbook" (sometimes referred to herein as the "Handbook").

The development methodology employed by SAI in achieving these objectives can be described at its highest level in five steps.

Step 1. Gain complete knowledge of the computer metrics technology developed by General Electric in April 1980 (RADC-TR-80-109, Vol. I & II).

Step 2. Employ a stepwise refinement to decompose the GE metric system to successively increasing levels of detail until all fundamental units, the "Data Elements", are described and understood.

Step 3. Evaluate each "Data Element" to identify those suitable for the "Handbook".

Step 4. Rebuild the system from the bottom up using only those "Data Elements" selected in Step 3.

Step 5. Present the metric system in the form of a "Practical Handbook".

As illustrated in Figure III-1, the "Framework of the Metrics Handbook" has the following five components: (1) General Instructions, (2) Quality Factor Selection Instructions, (3) Data Element Dictionary, (4) Quality Factor Modules (eleven), and

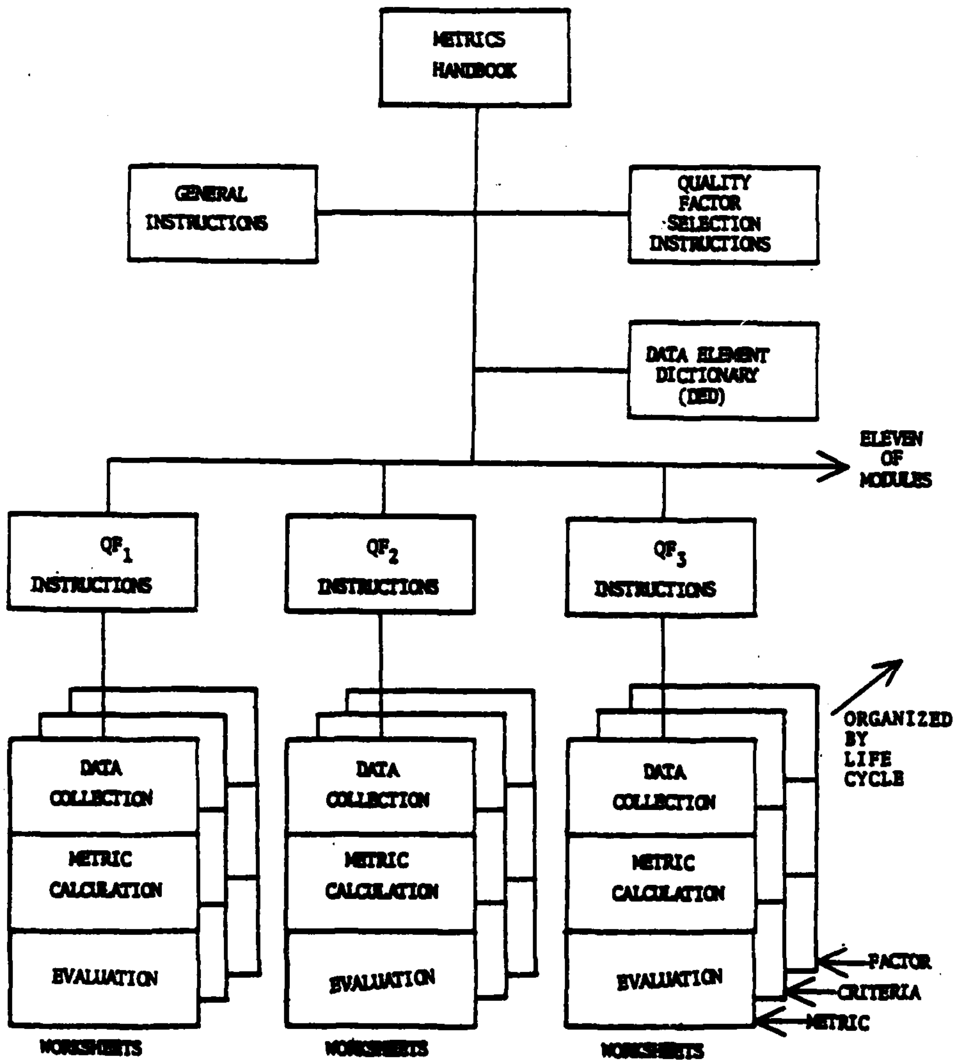


FIGURE III-1

FRAMEWORK OF THE METRICS HANDBOOK

(5) Sets of Data Collection, Metrics, Evaluation Worksheets, Criteria for metric and Factor level organized according to the life cycle model. The procedure for applying the "Handbook" is hierarchical and can be described at its highest level in four steps;

Step 1. Decide whether the handbook is suitable for the system under development based on the "General Instructions".

Step 2. Select the Quality Factors relevant to the system based on the "Quality Factor Selection Instructions".

Step 3. Obtain only those Quality Factor Modules that correspond to the Quality Factors selected in Step 2.

Step 4. Apply the worksheets repeatedly over the system life cycle as described in the "Quality Factor Module Instructions".

This framework of procedure provides a software quality measurement system which is quantitative in value, yet simple enough to be learned in one week by a student with limited software development background.

A sample worksheet from the Handbook is included on the following page. In the upper right hand corner of each worksheet is the Form Code. Each worksheet is assigned a Form Code according to the Form Code Key below in Table III-A. When the worksheets are organized according to Form Code, the eleven "Quality Factor Modules" are formed.

DESIGN STRUCTURE MEASURE

Form Code: RePDM.4

LIFE CYCLE PHASE:

PRELIMINARY DESIGN

- [ ] SYSTEM
- [ ] SUBSYSTEM
- [ ] MODULE

NAME:__________

SOURCE(S):

	SCORE
I. DATA COLLECTION WORKSECTION:	
1.0 Hierarchical Structure (35)	
1.1 Is a hierarchical chart provided which identifies all modules in the system? [Y|N] Yes = 1, No = 0	☐
2.0 Module Independence (36)	
2.1 Is the module independent of the source of the input or the destination of the output? [Y|N] Yes = 1, No = 0	☐
3.0 Size of Data Base (70)	
3.1 Number of unique data items in data base. ☐ Score = 1 ÷ $\boxed{3.1}$	☐

II. METRIC WORKSECTION:

$$\text{Metric Value} = \frac{\text{Sum of Above Scores}}{\text{No. of Data Elements}}$$ ☐

III. EVALUATION WORKSECTION: What is your evaluation of the reviewed products based on the metrics above? (1-10)____ (Ø If you are unable to evaluate)

IV. INSPECTOR'S COMMENTS:

PREPARED BY:__________ APPROVED BY:__________

DATE:__________ DATE:__________

MODULE PHASE LEVEL SEQUENCE

ItRAC.2

INTEGRITY REQ-ANA. CRITERIA SECOND PAGE OF SET

MODULE	PHASE	LEVEL
Co = CORRECTNESS	RA = REQUIREMENTS ANALYSIS	M = METRIC
Re = RELIABILITY	PD = PRELIMINARY DESIGN	C = CRITERIA
Ef = EFFICIENCY	DD = DETAIL DESIGN	F = FACTOR
It = INTEGRITY	IM = IMPLEMENTATION	
Us = USABILITY		
Ma = MAINTAINABILITY		
Fx = FLEXIBILITY		
Te = TESTABILITY		
Po = PORTABILITY		
Ru = REUSABILITY		
Ip = INTEROPERABILITY		

TABLE III-A

FORM CODE KEY

The following development methodology described in the balance of this section expands on the evaluation of each "Data Element" in the GE metric system to identify those suitable for the "Computer System Acquisition Metrics Handbook". Those selected are included in the "Worksheets" of the eleven "Quality Factor Modules".

3.2 EVALUATION CRITERIA FOR DEVELOPING THE COMPUTER SYSTEM ACQUISITION HANDBOOK

The SAI technical staff established evaluation criteria for selecting Data Elements to be used in the handbook. This was done so only suitable Data Elements would be included while obscure or difficult Data Elements would be excluded. The three criteria chosen for this purpose were (1) Period, (2) Training, and (3) Importance. The degree of significance of each data element was evaluated against each of the three criteria and the resulting score plotted in a three-dimensional space with the three criteria as unit vectors. Figure III-2 illustrates this concept. The three criteria were selected for their orthogonality and significance relative to the objective of the "Handbook". A three-dimensional space was chosen as a model to preserve the concept of orthogonality.

3.2.1 Period

The evaluation criterion Period is based on the amount of time in minutes it takes for the person making the measurements to acquire information asked for in the Data Element .

3.2.2 Importance

The evaluation criterion Importance is based on the number of times a Data Element occurs in the total number of measuremets and how effective it is in the related Metric algorithims.

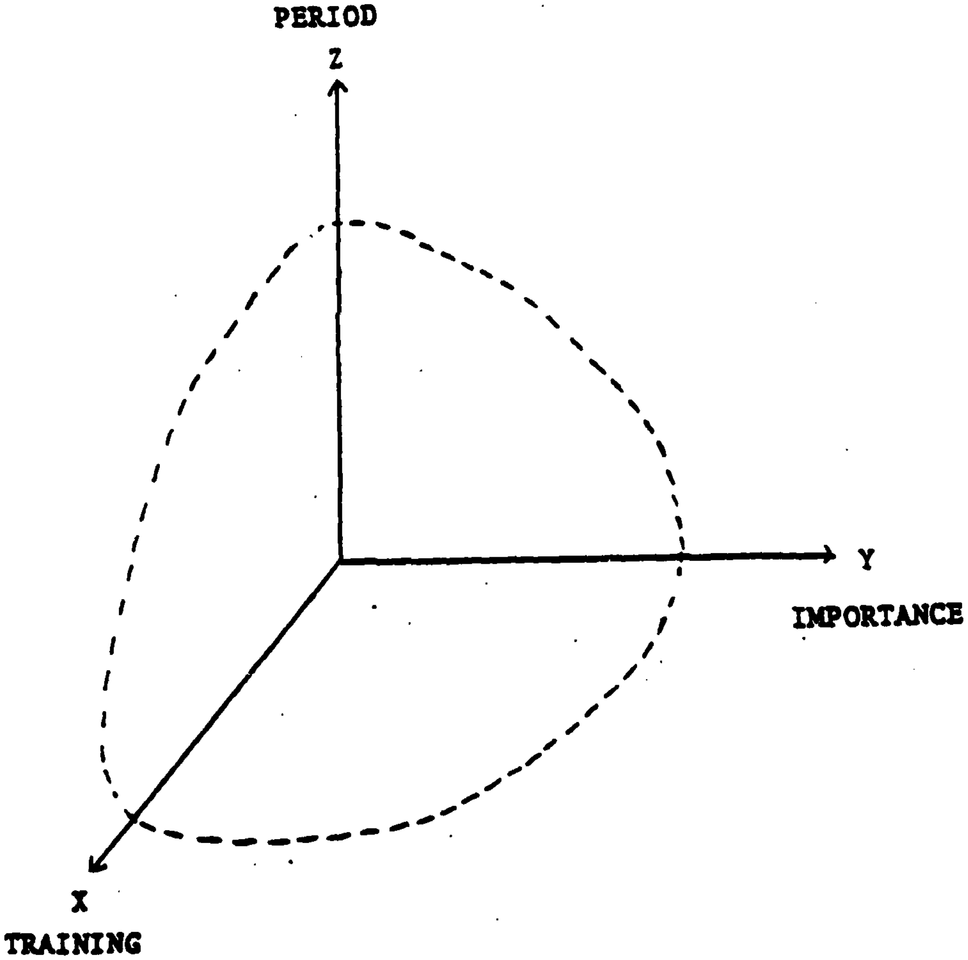


FIGURE III-2

THREE DIMENSIONAL EVALUATION SPACE

MODULE PHASE LEVEL SEQUENCE

ItRAC.2

INTEGRITY REQ-ANA. CRITERIA SECOND PAGE OF SET

MODULE	PHASE	LEVEL
Co = CORRECTNESS	RA = REQUIREMENTS ANALYSIS	M = METRIC
Re = RELIABILITY	PD = PRELIMINARY DESIGN	C = CRITERIA
Ef = EFFICIENCY	DD = DETAIL DESIGN	F = FACTOR
It = INTEGRITY	IM = IMPLEMENTATION	
Us = USABILITY		
Ma = MAINTAINABILITY		
Fx = FLEXIBILITY		
Te = TESTABILITY		
Po = PORTABILITY		
Ru = REUSABILITY		
Ip = INTEROPERABILITY		

TABLE III-A

FORM CODE KEY

The following development methodology described in the balance of this section expands on the evaluation of each "Data Element" in the GE metric system to identify those suitable for the "Computer System Acquisition Metrics Handbook". Those selected are included in the "Worksheets" of the eleven "Quality Factor Modules".

3.2 EVALUATION CRITERIA FOR DEVELOPING THE COMPUTER SYSTEM ACQUISITION HANDBOOK

The SAI technical staff established evaluation criteria for selecting Data Elements to be used in the handbook. This was done so only suitable Data Elements would be included while obscure or difficult Data Elements would be excluded. The three criteria chosen for this purpose were (1) Period, (2) Training, and (3) Importance. The degree of significance of each data element was evaluated against each of the three criteria and the resulting score plotted in a three-dimensional space with the three criteria as unit vectors. Figure III-2 illustrates this concept. The three criteria were selected for their orthogonality and significance relative to the objective of the "Handbook". A three-dimensional space was chosen as a model to preserve the concept of orthogonality.

3.2.1 Period

The evaluation criterion Period is based on the amount of time in minutes it takes for the person making the measurements to acquire information asked for in the Data Element .

3.2.2 Importance

The evaluation criterion Importance is based on the number of times a Data Element occurs in the total number of measurements and how effective it is in the related metric algorithims.

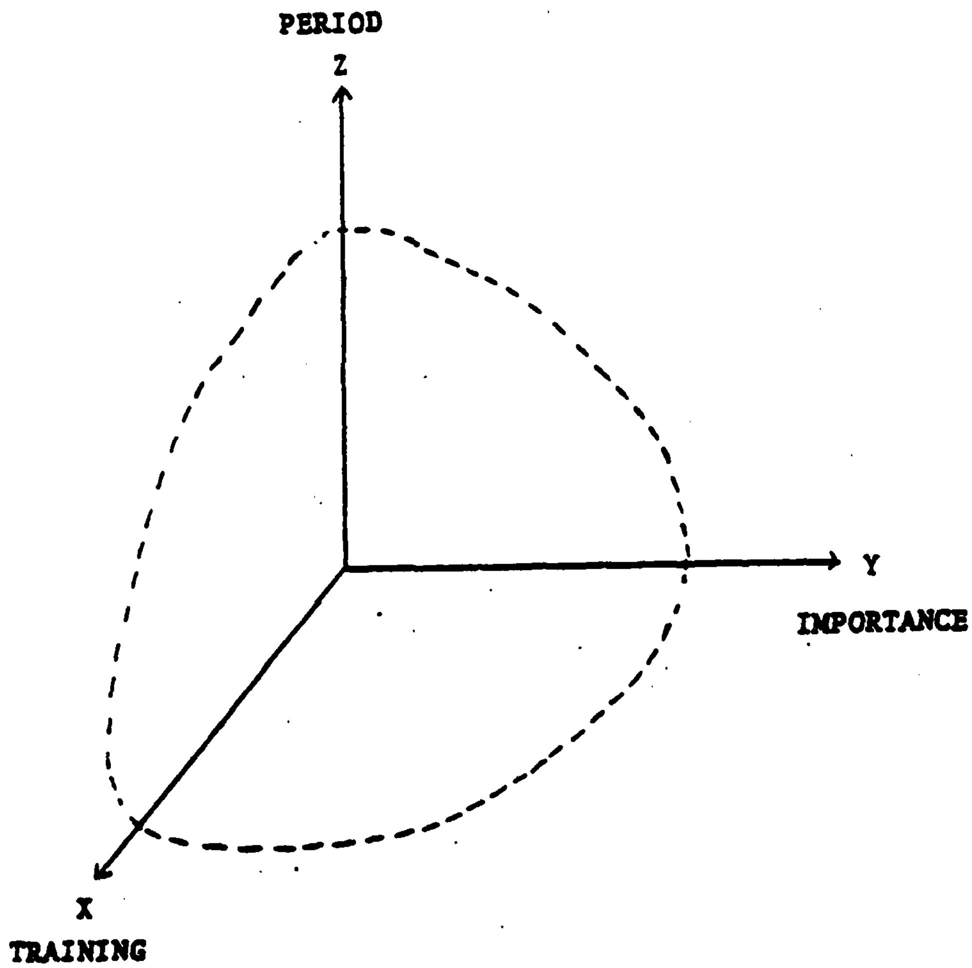


FIGURE III-2

THREE DIMENSIONAL EVALUATION SPACE

3.2.3 Training

The evaluation criterion Training is based on the amount of time it takes for a person to learn and understand a Data Element. This time estimate is based on the amount of software experience anticipated in the target audience.

3.3 THE DATA ELEMENTS FROM THE GE WORKSHEETS

Data Elements are a set of very specific questions about system/software characteristics over the development life cycle. These questions are the main initial materials for the "Handbook: development. SAI identified all the different Data Elements from the four GE Metrics worksheets. The Data Elements can be found in their original form on Metric Worksheets 1, 2a, 2b, and 3, pages 38 through 50 of RADC-TR-80-109, Vol. II, Final Technical Report, April 1980, Software Quality Measured Manual.

3.4 METHODOLOGY FOR EVALUATING DATA ELEMENTS SUITABLE FOR THE HANDBOOK

Once all the Data Elements were identified, the following evaluation was applied. Each Data Element was measured according to the three evaluation criteria described in Section 3.2, (Period, Importance, and Training). Each Data Element received a Period score, an Importance score, and a Training score. Each score ranges from one to five for each Data Element, and was represented by a point in the three-dimensional space of Figure III-3. Sections 3.4.1 through 3.4.3 describe the methodology that was employed to assign Period, Importance, and Training scores for the Data Elements.

3.4.1 Period Score

For the evaluation criterion Period, a Data Element was scored according to the approximate time in minutes it would take someone to acquire information asked for by that Data Element. The score has a range of 1 to 5, each score corresponding to a period of time in minutes. The less time that is involved, the higher the score. Figure III-4 illustrates the Period algorithm.

3.4.2 Importance Score

For the evaluation criterion Importance, a Data Element was scored according to (1) the number of occurrences in the metric system, and (2) its effectiveness.

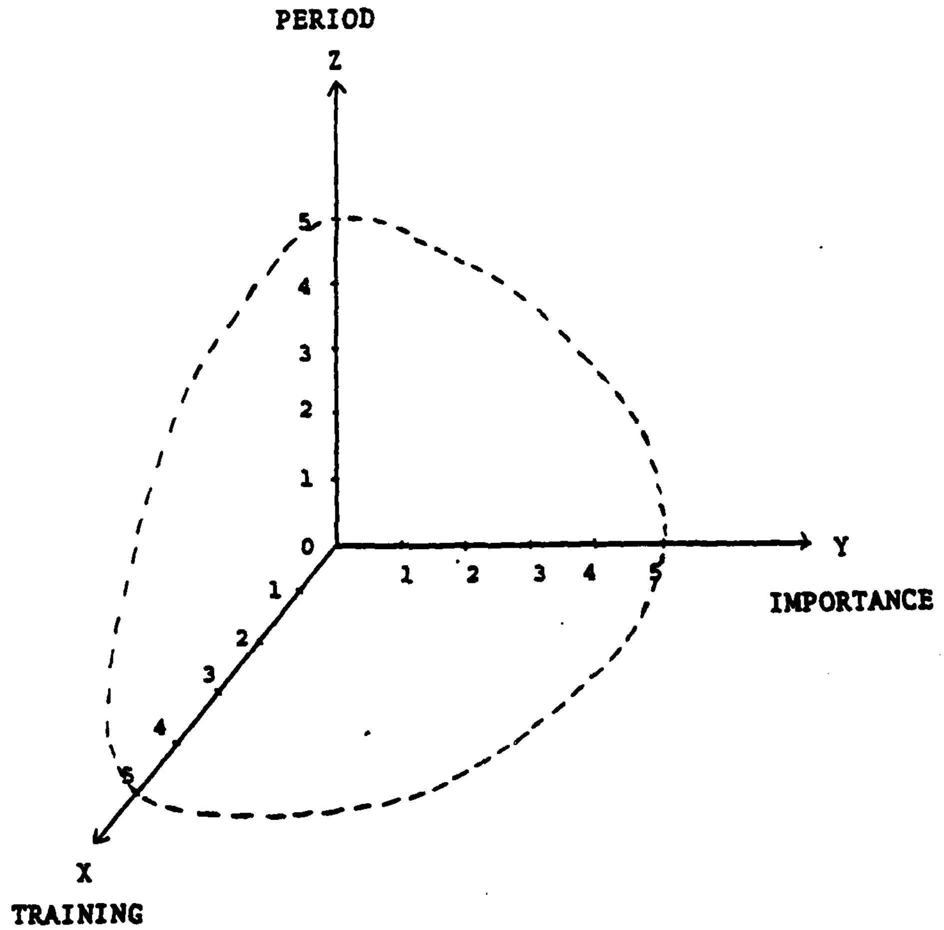
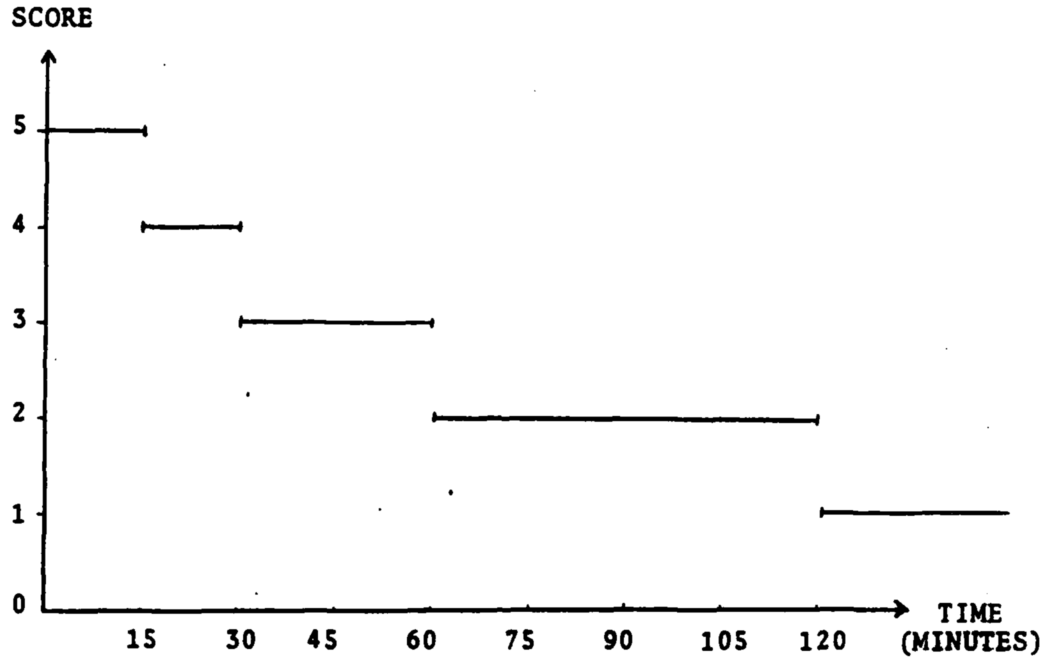


FIGURE III-3

THREE DIMENSIONAL EVALUATION SPACE

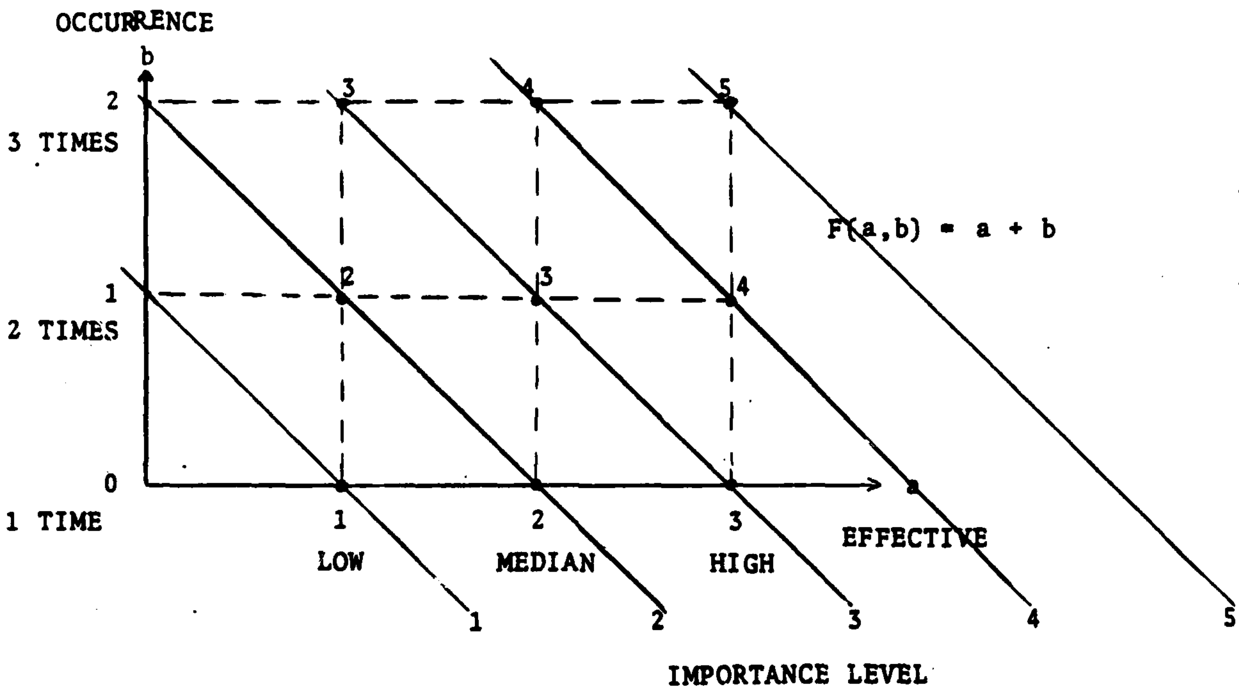
within the related metric algorithm. The Importance score is acquired by adding the Occurrence and Effective scores. Figure III-5 illustrates the Importance Algorithm.



SCORE	TIME (MINUTES)
5	0- 15
4	15- 30
3	30- 60
2	60-120
1	OVER 120

FIGURE III-4

PERIOD ALGORITHM



b :

OCCURRENCE	1 TIME	2 TIMES	3 TIMES
SCORE	0	1	2

a:

EFFECTIVE	LOW	MEDIAN	HIGH
SCORE	1	2	3

F(a,b):

OCCURRENCE	2	1	2	2	1	0	1	0	0
EFFECTIVE	3	3	2	1	2	3	1	2	1
SCORE	5	4		3			2		1

FIGURE III-5

IMPORTANCE ALGORITHM

3.4.3 Training Score

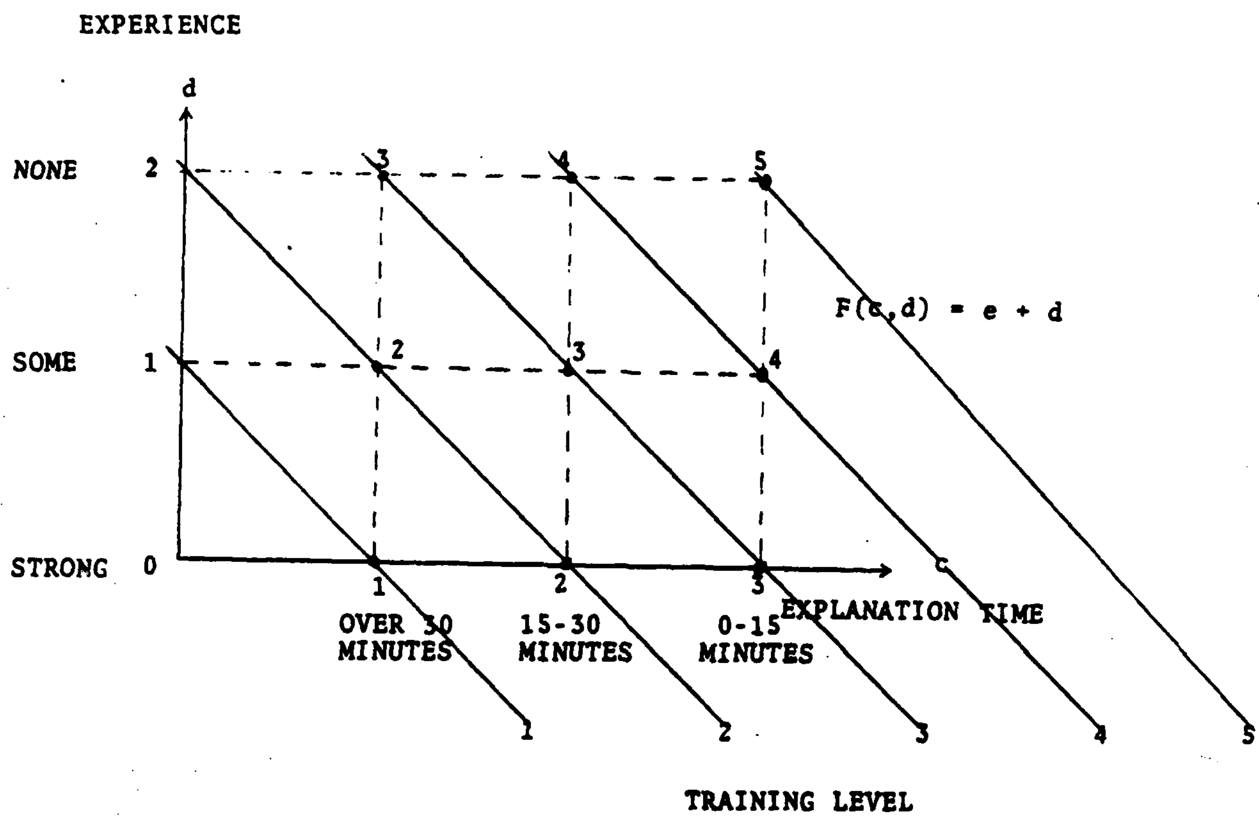
For the evaluation criterion, Training, a Data Element was scored according to: (1) The approximate time in minutes it would take someone to understand a Data Element (Explanation Time); and (2) The amount of related experience that person had (Experience). A short explanation time scored high and a low experience scored high. The Training score is obtained by adding the Explanation Time and Experience scores. Figure III-6 illustrates the Training Algorithm.

3.5 SELECTING DATA ELEMENTS SUITABLE FOR THE HANDBOOK

After assigning a score in each of the three dimensions for each Data Element, the next step was to select a method of combining the three scores to produce a composit score for each Data Element. The purpose was to give each Data Element a score based on all three evaluation criteria. This was applied to all Data Elements. Once all the Data Elements were represented by one score, a score distribution analysis was conducted. This classified the Data Elements into three categories: (1) Keep; (2) Reserve; and (3) Drop. The following subsections describe this whole process in detail.

3.5.1 Candidate Selection Methodologies

SAI considered three composit scoring methods for the Data Elements: (1) Sum; (2) Vector Length; and (3) Product. The Data Element score in the Sum method is derived from the sum of the Data Element's three evaluation criteria scores. The Data Element score in the Vector Length method is derived from the square root of the sum of the squared criteria scores. The Data Element score in the Product method is derived from the product of the criteria scores.



d:

EXPERIENCE	STRONG	SOME	NONE
SCORE	0	1	2

c:

EXPLANATION TIME	OVER 30 MINUTES	15-30 MINUTES	0-15 MINUTES
SCORE	1	2	3

F(c,d):

EXPERINCE	2	2	1	2	1	0	1	0	0
EXPLANATION TIME	3	2	3	1	2	3	1	2	1
SCORE	5	4		3			2		1

FIGURE III-6

TRAINING ALGORITHM

3.5.2 Comparison of Selection Methodologies

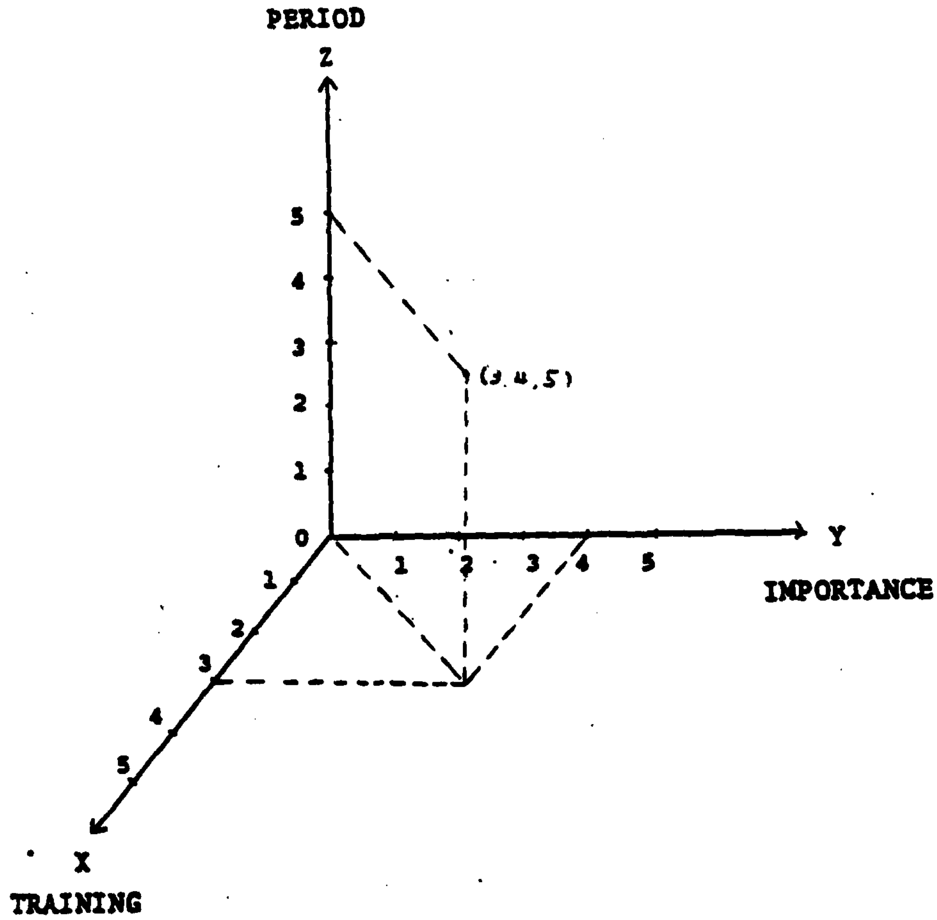
SAI tested each candidate selection methodology on all the Data Elements to determine the most efficient one for the Data Element selection process. Figure III-7 illustrates examples of the three methods on one Data Element that has a Period score of 5, an Importance score of 4, and a Training score of 3.

3.5.3 Best Selection Methodology

SAI chose the Sum method as the best selection methodology because the distribution of scores derived from it was smooth and clear compared to the other methods. Using the sum method as a base of scores for the Data Elements, SAI classified each of the Data Elements into a population of three classes: (1) Keep; (2) Reserve; and (3) Drop. The mean and standard deviation of the Data Element score distribution establishes the Keep, Reserve, and Drop categories. Figure III-8 illustrates the distribution. Twenty Data Elements with low scores were dropped; 33 with higher scores were put on Reserve, while the rest were kept.

3.5.4 Comparison of GE and SAI Data Element Evaluations

The next step was to compare SAI and GE scoring criteria to see if the ratings they gave for the Data Elements were significantly different. Figures III-9 and III-10 show the scoring criteria utilized by GE and SAI. Notice that the SAI criterion time is equivalent to GE's Criterion Effort and SAI's Training Requirement is equivalent to GE's Skill Level.



PRODUCT METHOD

$F(X,Y,Z) = X \times Y \times Z$

EXAMPLE: WHEN X=3, Y=4, Z=5, THEN

$F(X,Y,Z) = F(3,4,5) = 3 \times 4 \times 5 = 60$

SUM METHOD

$F(X,Y,Z) = X + Y + Z$

EXAMPLE: WHEN X=3, Y=4, Z=5, THEN

$F(X,Y,Z) = F(3,4,5) = 3+4+5 = 12$

VECTOR LENGTH METHOD

$F(X,Y,Z) = (X^2 + Y^2 + Z^2)^{1/2}$

EXAMPLE: WHEN X=3, Y=4, Z=5, THEN

$F(X,Y,Z) = F(3,4,5) = (3^2 + 4^2 + 5^2)^{1/2} = 7.1$

FIGURE III-7

EXAMPLE OF THE SCORING OF A DATA ELEMENT USING THREE DIFFERENT METHODOLOGIES

SCORE OF SUM	15	14	13	12	11	10	9	8	7	6	5	4	3
NUMBER	1	1	3	6	21	23	25	13	23	10	10	7	3

	SCORE	NUMBER	%
KEEP	15-8	93	64%
RESERVE	7-6	33	23%
DROP	5-3	20	13%

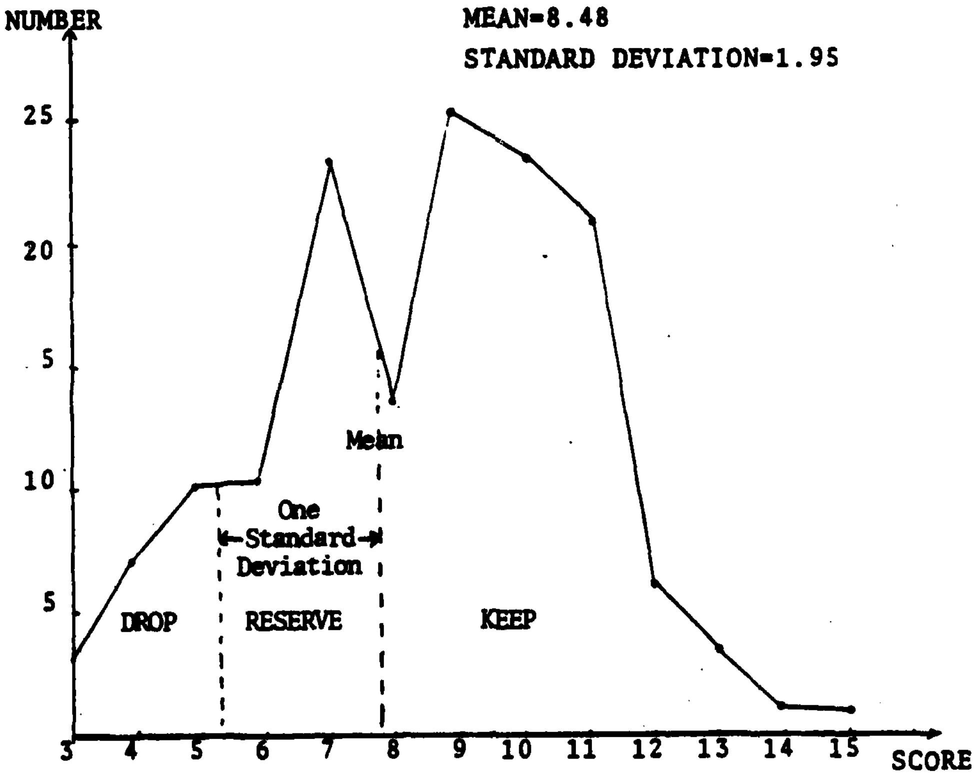


FIGURE III-8

SCORE DISTRIBUTION FOR SUM METHCD

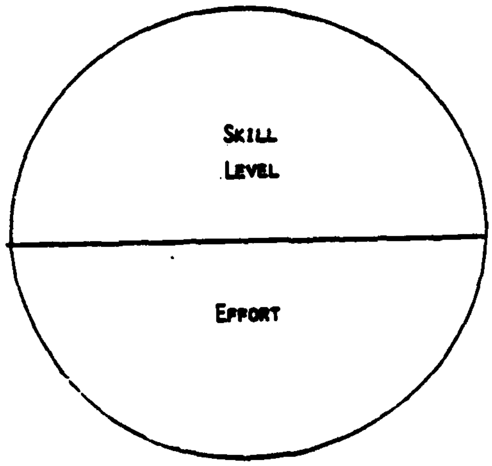


FIGURE III- 9

GE SCORE CRITERIA

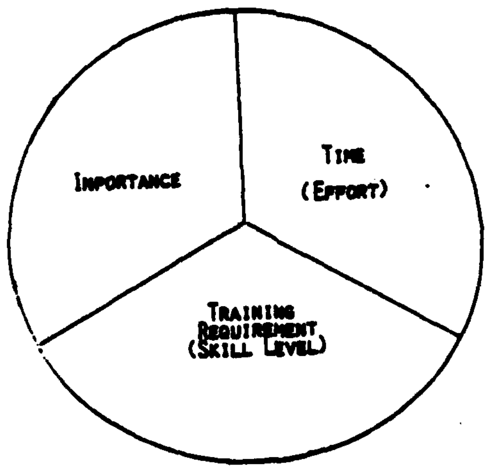


FIGURE III-10

SAI SCORE CRITERIA

Figure III-11 shows the comparison between the two scoring methodologies. GE's Skill Level and Effort scores range from 1 (lowest) to 5 (highest) while SAI's scores are the exact opposite. Both GE and SAI score criteria were used to measure the Data Elements. Table III-B shows a score distribution, average score and results of applying Skill Level measurements to the Data Elements. Table III-C shows the same type of analysis but with the Effort measurement, while Table III-D shows both criteria combined. The figures show that there was a slight difference in the scoring results but not a significant one.

3.5.5 Impact of Reducing Number of Data Elements on Quality Factors

It was important to assess what "dropping" data would do at higher levels. Would dropping a set of Data Elements be equivalent to eliminating a Quality Factor?

As Table III-E shows, dropping 20 Data Elements resulted in the elimination of two metrics from the system. Clearly, there was no significant impact on any of the 11 Quality Factors.

3.5.6 Categories of Data Elements

SAI classified 146 GE Data Elements before the selection process described above. Of these, 93 fell into the Keep Category, 33 fell into the Reserve Category, and 20 fell into the Drop Category. Tables III-F to III-H list the Data Elements according to category along with the score under all these selection methodologies. The Data Element listed by the SAI sequence code (numerical value) and the GE code, as defined in RADC-TR-80-109, are in the left columns of the tables. The columns to the right show the scores of the data elements derived from the Sum, Vector Length, and Product Scoring methodologies.

GE (S)		SAI (S)
SCORE		SCORE
1	CLERK	5
2	ENTRY LEVEL PROGRAMMER	4
3	EXPERIENCED PROGRAMMER	3
4	MATH ANALYST	2
5	SYSTEM ANALYST	1

WE REVERSED THE SCORES SUPPLIED BY GE TO BE IN LINE WITH THE SCORING METHODOLOGY OF SAI.

(E)		(E)
1	MINIMUM	5
2	MINOR	4
3	MODERATE	3
4	MAJOR	2
5	MAXIMUM	1

FIGURE III-11

SCORING METHODOLOGY
GE AND SAI

Skill(s) Level	5	4	3	2	1	Total	Avg Score	Avg. Ability
SAI	15	11	75	18	27	146	2.79	Need more ability than an experienced Programmer
GE	23	73	37	1	0	140	3.93	Entry Level Programmer

Low Skill ⟶ High Skill
5 1

TABLE III-B
SKILL LEVEL

Effort to (E) Collect	5	4	3	2	1	Total	Avg Score	Avg. Effort
SAI	13	47	50	20	16	146	3.14	Moderate Amount
GE	75	55	8	2	0	140	4.45	Between very little and little

Minimum Effort → Maximum Effort

5 1

TABLE III- C

EFFORT LEVEL

(S) & (E)	10	9	8	7	6	5	4	3	2	TOTAL	AVG. SCORE
SAI	2	5	26	28	33	20	12	10	10	146	2.97
GE	16	50	49	24	0	1	0	0	0	140	3.19

LOW → HIGH

5 → 1

TABLE III-D

COMBINED SKILL AND EFFORT LEVEL

		SELECTION		
	ORIGIN	KEEP	RESERVE	DROP
FACTOR	11	11	0	0
CRITERIA	22	22	0	0
METRIC	39	33	4	2
DATA ELEMENT	146	93	33	20

TABLE III-E

DATA ELEMENT SELECTION STATISTICS

3.6 MAPPING OF G.E. METRICS INTO SOFTWARE METRICS USED IN THIS HANDBOOK

All 126 data elements in the Keep and Reserve categories were selected for the Metrics handbook. The ones in the Drop category were not included. Tables III-F and III-G list the data elements that were suitable for the "Handbook". They were incorporated into the worksheets according to the Framework described in Subsection 3.1.

The bottom-up approach, going from data element to metric to criteria and finally Quality Factor according to the original framework was observed and produces the modules of this handbook.

#	Sum	Vector Length	Product	#	Sum	Vector Length	Product
1 CP.1	15	8.7	125	33 ET.4(2)	11	6.6	45
54 TN.1(1)	14	8.1	100	11 ET.2(1)	11	6.6	45
3 CP.1(2)	13	7.7	75	17 AC.1(3)	10	6.5	20
4 CP.1(3)	13	7.7	75	117 SD.1	11	6.5	45
5 CP.1(4)	13	7.7	75	118 SD.2(1)	11	6.5	45
15 AC.1(1)	12	7.1	60	119 SD.2(3)	11	6.5	45
16 AC.1(2)	12	7.1	60	120 SD.2(4)	11	6.5	45
55 TN.1(2)	12	7.1	60	121 SD.2(5)	11	6.5	45
133 AY.1(5)	12	7.1	60	122 SD.2(6)	11	6.5	45
8 CP.1(7)	12	7.1	60	123 SD.3(2)	11	6.5	45
1 CP.1(1)	12	6.9	64	124 SD.2(7)	11	6.5	45
126 SD.3(4)	11	6.7	40	20 OP.1(1)	11	6.4	40
97 MI.1(1)	11	6.7	40	19 AA.1(2)	11	6.4	48
96 SS.1(2)	11	6.7	40	143 CM.2(4)	11	6.4	48
57 CM.1(2)	11	6.6	45	142 CM.2(3)	11	6.4	48
56 TN.1(3)	11	6.6	45	141 CM.2(2)	11	6.4	48
34 ET.5(2)	11	6.6	45	24 OP.1(3)	10	6.0	32

TABLE III-F

DATA ELEMENTS IN THE KEEP CATEGORY

#	Sum	Vector Length	Product	#	Sum	Vector Length	Product
18 AA.1(1)	10	6.0	32	140 OP.1(7)	10	5.8	36
125 SD.3(3)	10	6.0	32	62 OM.2(5)	10	5.8	36
137 SD.2(2)	10	6.0	32	63 OM.2(6)	10	5.8	36
136 EE.3(1)	10	6.0	32	93 GE.2(2)	10	5.8	36
6 CP.1(5)	10	5.8	36	107 SD.3(1)	9	5.7	16
10 AY.1(2)	10	5.8	36	132 MI.1(4)	9	5.4	24
12 ET.3(1)	10	5.8	36	105 CS.2(2)	9	5.4	24
13 ET.4(1)	10	5.8	36	104 CS.1(4)	9	5.4	24
14 ET.5(1)	10	5.8	36	103 CS.1(2)	9	5.4	24
23 OM.1(6)	10	5.8	36	102 CS.1(3)	9	5.4	24
24 OM.2(7)	10	5.8	36	101 CS.1(1)	9	5.4	24
25 OM.2(1)	10	5.8	36	100 EE.2(1)	9	5.4	24
26 EE.1	10	5.8	36	95 GE.2(4)	9	5.4	24
43 OP.1(6)	10	5.8	36	94 GE.2(3)	9	5.4	24
48 OM.1(1)	10	5.8	36	28 DC.1(1)	9	5.4	24
49 OM.1(3)	10	5.8	36	27 CC.1(1)	9	5.4	24
60 OM.1(4)	10	5.8	36	21 OP.1(2)	9	5.4	24

TABLE III-F (cont'd)

DATA ELEMENTS IN THE KEEP CATEGORY

#	Sum	Vector Length	Product	#	Sum	Vector Length	Product
80 ET.3(4)	9	5.2	27	29 TR.1	8	5.1	12
79 ET.3(3)	9	5.2	27	146 SE.1(6)	8	5.1	12
78 ET.3(2)	9	5.2	27	64 IN.1(1)	8	4.7	18
77 ET.2(4)	9	5.2	27	145 EE.2(3)	8	4.7	18
75 ET.2(2)	9	5.2	27	46 CC.1(2)	8	4.7	18
76 ET.2(5)	9	5.2	27	70 SI.1(6)	8	4.7	18
67 IN.2(2)	9	5.2	27	78 IN.3(1)	8	4.7	18
66 IN.2(1)	9	5.2	27	49 DC.1(2)	8	4.7	18
138 SI.1(4)	9	5.2	27	9 AY.1(1)	8	4.7	18
7 CD.1(6)	9	5.2	27	82 SI.1(3)	8	4.7	18
61 CM.1(5)	9	5.2	27	83 SI.1(5)	8	4.7	18
32 ET.1(2)	9	5.2	27	106 SI.4(2)	8	4.7	18
35 SI.1(1)	8	5.1	12				

TABLE III-F (cont'd)

DATA ELEMENTS IN THE KEEP CATEGORY

#	Sum	Vector	Product	#	Sum	Vector	Product
99 EX.2(2)	7	4.6	8	110 SI.4(8)	7	4.1	9
31 ET.1(1)	7	4.6	8	111 SI.4(9)	7	4.1	9
38 SE.1(1)	7	4.6	8	112 SI.4(3)	7	4.1	9
39 SE.1(2)	7	4.6	8	115 SI.4(1)	7	4.1	9
98 EX.1(1)	7	4.4	9	144 CS.2(1)	7	4.1	9
73 AY.1(4)	7	4.1	12	113 SI.4(4)	7	4.1	9
72 ET.1(3)	7	4.1	12	135 SE.1(4)	6	3.7	6
65 IN.1(2)	7	4.1	12	128 SI.4(10)	6	3.7	6
74 ET.2(2)	7	4.1	12	91 MO.2(7)	6	3.7	6
85 SS.1(1)	7	4.1	12	81 SI.3	6	3.7	6
83 MO.2(3)	7	4.1	12	69 IN.3(2)	6	3.7	6
88 MO.2(4)	7	4.1	12	50 DC.1(3)	6	3.7	6
89 MO.2(5)	7	4.1	12	44 EE.2(5)	6	3.7	6
90 MO.2(6)	7	4.1	12	43 EE.3(5)	6	3.7	6
92 GE.2(1)	7	4.1	12	42 CS.2(3)	6	3.7	6
108 SI.4(6)	7	4.1	9	36 SI.1(2)	6	3.7	6
109 SI.4(7)	7	4.1	9				

TABLE III-G

DATA ELEMENTS IN THE RESERVE CATEGORY

#	Sum	Vector	Product	#	Sum	Vector	Product
37 GE.1	5	3.3	3	131 MI.1(3)	4	2.4	2
45 EE.2(4)	5	3.3	3	130 EE.3(2)	4	2.4	2
47 CC.1(3)	5	3.3	3	114 SI.4(5)	4	2.4	2
48 CC.1(4)	5	3.3	3	84 EX.2(1)	4	2.4	2
51 OP.1(4)	5	3.3	3	71 SI.1(7)	4	2.4	2
52 OP.1(5)	5	3.3	3	40 SE.1(5)	4	2.4	2
86 MI.1(2)	5	3.3	3	30 AY.1(3)	4	2.4	2
129 SE.1(3)	5	3.3	3	41 SE.1(7)	3	1.7	1
127 MO.2(1)	5	3.0	4	116 CO.1	3	1.7	1
134 EX.2(3)	5	3.0	4	139 EX.1(2)	3	1.7	1

TABLE III-H

DATA ELEMENTS IN THE DROP CATEGORY

SECTION IV

PILOT APPLICATION OF SOFTWARE METRICS HANDBOOK

4.1 TRAINING PHASE OF SOFTWARE METRICS HANDBOOK

A training course for ESD/TOEE personnel on the metrics concept and use of the "handbook" procedures was held at ESD/TOEE during the last phase of the contract. The purpose of this training course was to demonstrate that the Software Metrics "handbook" can easily be learned by inexperienced personnel. The materials developed for this course and modified as a result of this training phase include: (1) Software Metrics Handbook; (2) Quality Factor Module Instructions; (3) Quality Factor Modules; (4) Data Element Dictionary; (5) Instructor's Guide; (6) Course Outline; and related training material and transparencies. These materials will allow ESD/TOEE to instruct their personnel in the application of the Software Metrics Handbook.

4.2 APPLICATION OF SOFTWARE METRICS HANDBOOK TO A C^3 SYSTEM

The Software Metrics Handbook was applied to an actual C^3 system by ESD/TOEE personnel working with SAI personnel. The system, provided by ESD, was the RADAR Prediction System (RAPS). The documentation for this system consisted of the System Specification, Development Specifications for various subsystems, Product Specifications for various subsystems and Source Code. The Metrics were applied on six separate occasions, the purpose being two-fold: To validate the applicability of the Software Metrics Handbook and to improve the Handbook components. This application resulted in the "fine tuning" of the Software Metrics Handbook.

4.3 ISSUES AND RESOLUTIONS

During the application of the Software Metrics Handbook several issues were raised regarding the Handbook components. These issues and resolutions are as follows:

Issue #1: The Quality Factor Selection method was too proceduralized.

Resolution #1: The General Instructions were modified to deproceduralize the selection method.

Issue #2: Certain information required by the Quality Factor selection forms was deemed unnecessary.

Resolution #2: This information requirement was removed from the Quality Factor Selection Survey Form.

Issue #3: Information should be included regarding cost, time, and scoring f Software Metrics during the tradeoff process.

Resolution #3: These issues are addressed in the training materials prepared for ESD to teach the use of the Handbook.

Issue #4: The flow of information between the General Instructions and the Module Instructions should be clarified.

Resolution #4: This clarification was achieved by modifying Section IV of the General Instructions - "How The Handbook Works".

Issue 5: The definition of "Module" must be clarified and the concepts of and applications of system level as opposed to module level should be discussed.

Resolution #5: These clarifications and discussions are included in the General Instructions, Section I - "Software Metrics".

Issue #6: More information must be supplied regarding the correct sequence for applying the worksheets.

Resolution #6: This information is provided in the General Instructions, Section I - "Software Metrics".

Issue #7: The section labeled, "Source" on the worksheets should be filled in by the person using it with the name of the system.

Resolution #7: The worksheet format was changed and instructions included in the General Instructions.

Issue #8: The concepts of granularity and subjectivity regarding the data elements need to be addressed in the Theoretical Supplement.

Resolution #8: Included in Section 5.2 of the Theoretical Supplement.

Issue #9: Blocks marked "System Level", "Module Level", and "Subsystem Level" should be added to the worksheets.

Resolution #9: The worksheets were modified to reflect this change.

Issue #10: Products may not map uniquely into a single life cycle phase. As a result, information referenced in preceding phases may be required to apply Software Metrics at a current phase.

Resolution #10: This issue is resolved in the General Instructions.

Issue #11: Emphasis should be placed on the content of products and not the name or type of products when applying the Software Metrics to system documentation.

Resolution #11: This issue is resolved in the General Instructions.

Issue #12: The Metrics "Training Checklist" in the Preliminary Design phase of the Quality Factor Module Usability should be included in the Requirements Analysis phase and deleted from the Prelimanary Design phase.

Resolution #12: The worksheets in the Usability Module were modified.

Issue #13: There is a lack of information regarding standards and conventions etc. in the Detail Design Phase.

Resolution #13: This issue was resolved by including the Computer Program Development Plan (CPDP) in the product tables for the eleven Quality Factor Modules, and including this information in the General Instructions, Section II - "Life Cycle Considerations".

Issue #14: The Quality Factor Modules may not be easily reproduced for use by ESD personnel.

Resolution #14: The masters will be delivered to ESD unbound to facilitate making copies.

Issue #15: All modules may not be applicable to all Computer Program Configuration Items (CPCI) or not applicable due to system characteristics.

Resolution #15: The General Instructions were expanded to include this information.

Issue #16: The Interpretation Worksection should be changed to evaluate the reviewable products on a 1 to 10 scale subjectively.

Resolution #16: The worksection was revised to contain the following question: "What is your opinion of the reviewed products, based on the data elements (or metrics or criteria) above?" (1-10) - (Ø if you have no opinion). More detail is provided in this Theoretical Supplement in Section V.

Issue #17: There should be information provided explaining the concept of threshold values based on historical data.

Resolution #17: This information is found in this Theoretical Supplement in Section V and in Section IV of the General Instructions.

SECTION V

CONCLUSIONS AND RECOMMENDATIONS FOR FURTHER ACTION

5.1 ALTERNATIVE USES FOR SOFTWARE METRICS

Software Metrics is a set of procedures for measuring essential aspects of software systems. Software Metrics was principally designed to be used as a tool to measure certain qualities in a system under development through the calculation of Quality Factor scores which are ultimately evaluated against historical data. These measures of the Quality Factor of a system are to be used as "early-warning" signals to point out dificiencies as they develop, so that they can be corrected as easily and inexpensively as possible. However, it is not always possible to use Software Metrics in this way because of the changes in policy or unforseen events during a software development. Therefore, versatility was designed into Software Metrics in order to allow alternative utilizations of this tool.

SAI has identified six alternative methods of using Software Metrics; (1) Use as measure of the impact of revisions to software, (2) Use as a review tool to determine the appropriateness of redesign, (3) Use to perform retrospective analyses of existing systems, (4) Use as performance incentives based on Software Metrics scores, (5) Use to develop guidelines for inhouse development of software, and (6) Use to maintain control and visibility of software development. Each of these alternatives will be discussed further in the remainder of this section.

5.1.1 Software Metrics Used to Measure Software Revisions

When revisions are made to software, the impact of these revisions could be measured by applying Software Metrics to the revised software. If the same Quality

Factors are chosen for the revised application and scores of the original software have been maintained, the revised and original scores could be compared to determine if the revisions had any negative or positive effect on the software. However, if different qualities have been determined to be important, separate sets of Quality Factors could be applied to the revised software. In this case, these scores could be maintained as historical data for future reference and comparison.

5.1.2 Software Metrics Used as a Review Tool

When a redesign of some software is being considered, Software Metrics could be applied to the existing software to determine the weak areas in the design with respect to the newly desired characteristics. The redesign could be guided by the Quality Factor scores that are low with respect to the newly desired characteristics. For instance, if a particular piece of software was not originally written with maintainability in mind, the Maintainability Quality Factor Module could be applied to the software. If Maintainability scored low, then work to improve maintainability would be needed.

5.1.3 Software Metrics Used to Perform Retrospective Analyses

For systems that were developed without the Metrics being applied during development, Software Metrics could be applied after development is over to determine where in the development problems if any originated. For example, if it has been determined that a particular system is very difficult to maintain, the Maintainability Software Metrics could be applied to all of the development documentation to find the source of the maintainability problem. Alternatively, if a system is determined to be outstanding in a particular quality, the Software Metrics could be applied to that software and the results could be used as examples of good scores to compare against future development efforts.

5.1.4 Software Metrics Used as Performance Incentives

Software contractors could be given performance incentives based on Software Metric scores. Software would be evaluated based on Software Metric scores. This would provide incentive to the contractor to design the desired qualities into the software.

5.1.5 Software Metrics Used to Develop Guidelines

If measurement of software quality is not desired, or if historical figures are not yet available so that that type of evaluation is not possible, software could be designed and implemented following guidelines developed from the Data Elements of the Metrics Handbook. Instead of measuring the existence of Quality Factors, the Data Element questions can be used to develop guidelines that reflect good programming practice.

5.1.6 Software Metrics Used for Control and Visibility

Software Metrics provides a stuctured approach to control and observation of the development of software throughout its life cycle. If the software metric data is collected, but not calculated to produce metric scores, it provides control and visibility to the development. Software Metrics data tracks the development of the system via the products produced during the life cycle. Life Cycle control of the development is thus easily accomplished.

5.2 MANAGEMENT CONCERNS

The use of Software Metrics as a Quality Assurance tool requires continuing concern by management of two major issues. The first issue is the granularity and subjectivity of several of the Data Element questions. The second issue is the necessity to track and monitor the collection of scores and evaluations and the developments of a Metric Data Base.

Granularity and subjectivity of the Data elements can be decreased over time by developing standard guidelines for resolving each of the questions which are determined to lack granularity or are too subjective. A procedure for monitoring and resolving these issues should be developed.

The second issue that management must consider is the necessity to track and monitor both the collection of the scores and evaluations. Tracking and monitoring of the scores is necessary in order to develop a Metric data base. These scores can be used to compare future re-applications of Software Metrics to the same system, or in the devleopment of other systems. The data base can be used to compare history with current development efforts.

Each Metric, Criteria, and Factor worksheet contains an Evaluation Worksection. It is important to note that this evaluation is subjective on the part of the person applying the Software Metrics. For instance, at the Criteria level, the Evaluation Worksection asks: "What is your evaluation of the reviewed products based on the Metrics above? ________ (1-1Ø or Ø if you are unable to evaluate)." When a person is evaluating a Criteria it would be possible to have one Metric with a high score and another with a low score. The evaluator could decide that the Criteria should be evaluated fairly high (5-8) because in his opinion the low scoring Metric did not have much of an impact on the system.

Because of this subjective nature, a particular evaluator may consistantly evaluate high, or consistantly evaluate low. Some method of tracking and monitoring this type of scoring should be developed so that an analysis of the scores and scorer can be done as a means of giving a proper interpretation to the historical data in the data base.

5.3 CONCLUSIONS AND RECOMMENDATIONS

Software Metrics is a state-of-the-art tool, and as such is still in its infancy. Therefore, some of the objectivity desired by a quantification of software quality has not yet developed. Objectivity develops as guidelines are set and as threshold values are developed. Secondly Software Metrics are immediately valuable in their present subjective state because it can be used as a checklist for monitoring software throughout its acquisition life cycle. This makes Software Metrics easy to use, easy to teach, and not time consuming to apply. This is currently the most feasible use of the software metrics. Use as a guideline and for control and visibility as earlier discussed in 5.1.5 and 5.1.6, provide immediate value from their application.

SAI recommends four actions be taken by ESD to build a metric history: (1) Apply the Software Metrics to a wide variety of software development efforts. As systems reach the maintainence phase independent assessments of the quality of the system should be performed and compared to the quality evaluation as measured by metrics; (2) Establish a process to evaluate and update threshold values as they are developed; (3) Maintain a Software Metrics historical data base; and (4) Improve threshold value believability over time by comparing independent quality assessments with the Software Metric Data Base.

APPENDIX A

LIST OF WORKS CITED IN SOFTWARE METRICS LITERATURE REVIEW

[BAKF72] Baker, F.T.
"Chief Programmer Team Management of Production Programming". IBM Systems Journal, Vol. 11, No. 1, pp. 56-73, 1972.

[BOEB79b] Boehm, Barry W.
"Software Engineering - As It Is". Proceedings of the Fourth International Conference of Software Engineering: 11-21, September 1979.

[BROF74] Brooks, F.P. Jr.
The Mythical Man-Month: Essays on Software Engineering. Reading, MA: Addison-Wesley Publishing, 1974.

[CAVJ78] Cavano, Joseph P. and McCall, James A.
"A Framework for the Measurement of Software Quality". Performance Evaluation Review 7 (November 1978): 133-9.

[CHER80] Cheung, Roger C.
"A User-Oriented Software Reliability Model". IEEE Transactions on Software Engineering, Vol. SE-6 (March 1980): 118-25.

[CURB80b] Curtis, Bill.
"Measurement and Experimentation in Software Engineering". Proceedings of the IEEE: 1144-1157, September 1980.

[FREN77] French, N.
"Programmer Productivity Rising Too Slowly: Tanaka". Computerworld, Vol. 11, No. 32, 1977.

[GILT77] Gilb, Tom
Software Metrics. Cambridge, MA: Winthrop Publishers, Inc., 1977.

[GLAR79] Glass, Robert L.
Software Reliability Guidebook. Englewood Cliffs, NJ: Prentice Hall, Inc., 1979.

[JENR79] Jensen, Randall W. and Tonies, Charles C., eds.
Software Engineering. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1979

[KOSA75] Kossiakoff, A., et al.
"DOD Weapon Systems Software Management Study," Johns Hopkins University Applied Physics Laboratory Report SR 75-3, June 1975.

[KOSD74] Kosy, D.
"Air Force Command and Control Information Processing in the 1980s: Trends in Software Technology," RAND Report R-1012-PR, June 1974.

[LAMR80] Lamkey, Robert J. and Pavy, Curtis T.
Software Control During Development and Adquisition. Report No. AFIT-LSSR-62-80, Air Force Institute of Technology, Wright-Patterson AFB, OH, School of Systems and Logistics, AD-A039 329. June 1980.

[LIEB79] Lientz, B.P. and Swanson, E.B.
"Software Maintenance: A User/Management Tug-of-War". Data Management, April 1979, pp. 26-30.

[LITB80] Littlewood, Bev.
"Theories of Software Reliability: How Good Are They and How Can They Be Improved?" IEEE Transactions on Software Engineering, Vol. SE-6 (September 1980): 489-500.

[MARR80] Marsh, R.
"Air Force C^3 Technologies," SIGNAL, Vol. 3.5, No. 1, September 1980.

[MCCJ77a] McCall, J.; Richards, P.' and Walters, G.
Factors in Software Quality, 3 Vols. (A049014) (A049015) (A049055), RADC-TR-77-369, November 1977.

[MCCJ80b] McCall, J. and Matsumoto, M.
Software Quality Measurement Manual, RADC-TR-80-109, Vol. 2, April 1980. (ADA086986)

[MCCJ80c] McCall, J.
"An Assessment of Current Software Metric Research". IEEE Publication, No. 0531-6863/80/0000-0323, 1980.

[MCCT76] McCabe, Thomas J.
"A Complexity Measure." IEEE Transactions on Software Engineering, Vol. SE-2 (December 1976): 308-320

[MILH71] Mills, H.D.
"Top-Down Programming in Large Systems." Debugging Techniques in Large Systems. R. Rustin, Ed. Englewood Cliffs, N.J.: Prentice Hall, Inc., 1971.

[MUSJ80b] Musa, John D.
"The Measurement and Management of Software Reliability." Proceedings of the IEEE: 1131-1143, September 1980.

[MYEW78] Myers, W.
"The Need for Software Engineering." Computer, February 1978.

[NELR78] Nelson. R.
"Software Data Collection and Analysis." Draft Report, RADC, September 1978.

[PUTL79a] Putnam. Lawrence H. and Fitzsimmons, Ann.
"Estimating Software Costs." Datamation, September 1979, pp. 189-90+.

[PUTL79b] Putnam, Lawrence H. and Fitzsimmons, Ann.
"Estimating Software Costs." Datamation, October 1979, pp. 171-2+

[SHNB80] Shneiderman, Ben.
Software Psychology - Human Factors in Computer and Information Systems. Cambridge, MA: Winthrop Publishers, Inc., 1980.